IBM Bayesian Optimization Accelerator

1.1.0.1

Deployment Guide

Contents

Chapter 1. Deployment Guide

BOA deployment process

The BOA deployment process has three distinct phases. The three subsequent phases are installing host prerequisites, installing BOA, and configuring BOA.

- **Installing host prerequisites:**

  BOA depends on a set host prerequisite software product. Examples of these products include Docker Engine and IBM Spectrum LSF. You use the installation and configuration tools that are provided by each of these prerequisite products to set up the prerequisite software.

- **Installing BOA:**

  After you completed the setup of the host prerequisite products, then install BOA. During the BOA installation, a BOA installation directory on the host is created, and the Docker images that are associated with BOA into the local Docker registry on the host are loaded. The BOA installation program installs the BOA.

- **Configuring BOA:**

  After BOA is installed, configure the BOA system by using the BOA configuration tool. During the configuration process, a set of deployment files that are used to run the BOA software on the host are created.

  The BOA configuration tool configures an initial setup to establish a working system. The tool is also used to update the configuration settings.

Installing and Configuring Host Prerequisite Software

Its mandatory to install the prerequisite software products before installing BOA on any Power9 server. Prerequisites software installation is a one time set up.

Installing CUDA: Check OS and sestatus: check the OS and sestatus of the VM before installing CUDA, as BOA has been deployed and tested one below recommended OS:

```
   cat /etc/redhat-release
     Red Hat Enterprise Linux Server release 7.6 (Maipo)
  sestatus
    SELinux status:                 disabled
```

If the status is not disabled, edit the file `/etc/selinux/config` and change the parameter `SELINUX=disabled`, you must reboot the system after you have updated the file `/etc/selinux/config`.

Configure CUDA prereq repository:
Configure CUDA repository with the given command that will add repository to path “/etc/yum.repos.d/at.repo”

```
cat > /etc/yum.repos.d/at.repo << EOF
[advance-toolchain]
name=Advance Toolchain IBM FTP
baseurl=ftp://public.dhe.ibm.com/software/server/POWER/Linux/toolchain/at/redhat/RHEL7
failovermethod=priority
enabled=1
gpgcheck=0
gpgkey=ftp://public.dhe.ibm.com/software/server/POWER/Linux/toolchain/at/redhat/RHEL7/gpg-
pubkey-6976a827-5164221b
EOF
```

Once the repository added run yum command to install the packages.

```
yum -y install advance-toolchain-at12.0-devel advance-toolchain-at12.0-runtime
```

Before installing CUDA package user need to run below packages:

```
yum -y install pciutils environment-modules
 source /etc/profile.d/modules.sh
 module load at12.0
 yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
 yum clean all
 yum -y install dkms
wget http://mirror.centos.org/altarch/7/os/ppc64le/Packages/vulkan-
filesystem-1.1.97.0-1.el7.noarch.rpm
rpm -i vulkan-filesystem-1.1.97.0-1.el7.noarch.rpm
```

Downloading and installing the CUDA repo package:

- To download and install the CUDA package, run the following commands:

```
wget http://developer.download.nvidia.com/compute/cuda/10.2/Prod/local_installers/cuda-repo-
rhel7-10-2-local-10.2.89-440.33.01-1.0-1.ppc64le.rpm
```

```
rpm -i cuda-repo-rhel7-10-2-local-10.2.89-440.33.01-1.0-1.ppc64le.rpm
```

Installing CUDA:

- Install CUDA and enable the `nvidia-persistneced` so that every time you reboot the server the nvidia servers starts automatically. Run the following commands:

```
yum -y install cuda
```

```
systemctl enable nvidia-persistenced
```

Modifying udev rules:
Modify the udev rules in the `vi /lib/udev/rules.d/40-redhat.rules` file as displayed in the following example. Comment the lines as mentioned below:

```
# Memory hotadd request
#SUBSYSTEM!="memory", ACTION!="add", GOTO="memory_hotplug_end"
#PROGRAM="/bin/uname -p", RESULT=="s390*", GOTO="memory_hotplug_end"

#ENV{.state}="online"
#PROGRAM="/bin/systemd-detect-virt", RESULT=="none", ENV{.state}="online_movable"
#ATTR{state}=="offline", ATTR{state}="$env{.state}"

#LABEL="memory_hotplug_end"
```

Reboot the host system after installing and configuring the host prerequisites.

- To check the host system, run the following command after the host system restated:

```
systemctl status nvidia-persistenced
```

```
nvidia-smi
```

An output similar to the following screen is displayed:

```
[root@prh0706A2 etc]# nvidia-smi
Wed Sep 30 10:35:22 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 10.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000004:04:00.0 Off |                    0 |
| N/A   40C    P0    55W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2...  Off  | 00000004:05:00.0 Off |                    0 |
| N/A   41C    P0    55W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2...  Off  | 00000035:03:00.0 Off |                    0 |
| N/A   39C    P0    54W / 300W |      0MiB / 16160MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2...  Off  | 00000035:04:00.0 Off |                    0 |
| N/A   41C    P0    54W / 300W |      0MiB / 16160MiB |      4%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
[root@prh0706A2 etc]#
```

Installing Docker:

- To install Docker, add the below repository to `/etc/yum.repos.d`:

```
# cat > /etc/yum.repos.d/docker.repo << EOF
[docker]
name=Docker
baseurl=http://ftp.unicamp.br/pub/ppc64el/rhel/7/docker-ppc64el/
enabled=1
gpgcheck=0
EOF
```

```
# yum -y install docker-ce
# systemctl enable --now docker
# systemctl status docker
```

Docker will be installed at /var/lib/docker/ by default. If we aren't installing docker at this default location which should be avoided then we need to create a symbolic link of this custom docker installation directory to the default location. For example, If docker is installed at `/home/opt/docker/` then create a symbolic link by running the below command between this directory and the default directory, that is `/var/lib/docker/`:

```
ln -s /home/opt/docker/containers/* /var/lib/docker/containers
```

Firewall Settings:
Firewall rules must allow ports configured for BOA.

Spectrum LSF 10.1:
Spectrum LSF 10.1 is used in boa appliance as a workload manager. It provides a comprehensive set of intelligent, policy-driven scheduling features that enables full utilization of your compute infrastructure resources and ensure optimal application performance. LSF in boa appliance leverages docker container jobs and runs Bayesian Optimization jobs in docker container on demand.

LSF installation and configuration:

Install and configure LSF to support the docker container jobs, memory, and GPU related configuration.

- Unpack LSF 10.1 tar file:

```
tar -xvf lsf10.1_distro.tar.gz
cd ga
tar -xvf lsf10.1_lsfinstall_linux_ppc64le.tar.Z
cd lsf10.1_lsfinstall
```

- Update `install.config`:
  The `install.config` file contains options for LSF installation and configuration. Update the values for LSF_TOP, LSF_ADMINS, and LSF_CLUSTER_NAME elements as follows:

  Note: The value of the LSF_ADMINS element must be `lsfadmin`.

```
 LSF_TOP="/home/share/lsf"
LSF_ADMINS="lsfadmin"
LSF_CLUSTER_NAME="boa_appliance"
LSF_MASTER_LIST=<"Hostname of Master LSF">
LSF_ENTITLEMENT_FILE=<"Full path of the entitlement file">
LSF_TARDIR=<"Full path to the directory containing the LSF distribution tar files">
CONFIGURATION_TEMPLATE="DEFAULT"
ACCEPT_LICENSE="Y"
ENABLE_GPU="Y"
```

- Create user:
  Create the LSF administrator user `lsfadmin`.

```
useradd lsfadmin
passwd lsfadmin
```

- User Permissions:
  Provide permissions to `lsfadmin` to files and directory on the host. Make sure the `/home/lsfadmin` directory has owner as `lsfadmin`. If `lsfadmin` is not the directory owner, run the following command to make `lsfadmin` the directory owner:

```
chown -R lsfadmin:lsfadmin /home/lsfadmin
groupadd docker
usermod -aG docker lsfadmin
```

  Create a service in the `/etc/hosts` file to provide write access to the `lsfadmin` user. The following screen displays an example script to create a service to provide write access to `lsfadmin`:

```
cat > /etc/systemd/system/set-etchosts-acl.service << EOF
[Unit]
Description=set etc hosts write permission to lsfadmin service

[Service]
ExecStart=/usr/bin/setfacl -m u:lsfadmin:rw /etc/hosts

[Install]
WantedBy=default.target
EOF

systemctl enable set-etchosts-acl.service
systemctl start set-etchosts-acl.service
```

- Install LSF:
  Execute following commands to install LSF:

  1. `./lsfinstall -f install.config`

     Then enter 1. An output similar to the following screen is displayed after successful LSF installation:

```
lsfinstall is done.

To complete your LSF installation and get your
cluster "boa_appliance" up and running, follow the steps in
"/home/ashish/ga/lsf10.1_lsfinstall/lsf_getting_started.html".

After setting up your LSF server hosts and verifying
your cluster "boa_appliance" is running correctly,
see "/home/share/lsf/10.1/lsf_quick_admin.html"
to learn more about your new LSF cluster.

After installation, remember to bring your cluster up to date
by applying the latest fix pack from IBM Fix Central.
https://www.ibm.com/support/fixcentral/

Detailed steps for getting fixes from Fix Central, are in the
LSF installation guide on IBM Knowledge Center.
http://www.ibm.com/support/knowledgecenter/search/fix%20central?scope=SSWRJV
```

Figure 1. LSF installation success

2. `/home/share/lsf/10.1/install/hostsetup --top="/home/share/lsf" --boot="y"`

   An output similar to the following screen is displayed after successful host setup:

```
Logging installation sequence in /home/share/lsf/log/Install.log

------------------------------------------------------------
    L S F    H O S T S E T U P    U T I L I T Y
------------------------------------------------------------
This script sets up local host (LSF server, client or slave) environment.

Setting up LSF server host "master-lsf" ...
Checking LSF installation for host "master-lsf" ... Done
Installing LSF service scripts on host "master-lsf" ... Done
LSF service ports are defined in /home/share/lsf/conf/lsf.conf.
Checking LSF service ports definition on host "master-lsf" ... Done
You are installing IBM Spectrum LSF - 10.1.0.8 Standard Edition.

... Setting up LSF server host "master-lsf" is done
... LSF host setup is done.
```

Figure 2. Host setup

3. `. /home/share/lsf/conf/profile.lsf`

4. Fix Pack 1 Update bashrc for lsfadmin user.

   Log in as the lsfadmin user and update bashrc file by running the following command:

```
ssh lsfadmin@localhost
echo source /home/share/lsf/conf/profile.lsf >> ~/.bashrc
```

- Update `lsf.conf`:
  The `lsf.conf` is LSF configuration file that controls the operation of LSF. Ensure following configuration parameters are in `$LSF_ENVDIR/lsf.conf`:

```
LSF_UNIT_FOR_LIMITS=MB
LSF_PROCESS_TRACKING=Y
LSF_LINUX_CGROUP_ACCT=Y
LSB_RESOURCE_ENFORCE="gpu cpu memory"
LSB_ENABLE_HPC_ALLOCATION=Y
LSF_GPU_AUTOCONFIG=Y
LSB_GPU_NEW_SYNTAX=extend
LSF_RSH=ssh
LSF_ROOT_REX=local
LSB_GPU_REQ="mode=exclusive_process"
```

- Update `lsf.shared`:
  The `lsf.shared` file contains common definitions that are shared by all load sharing clusters defined by `lsf.cluster.cluster_name` files. Ensure the following configuration is in the `$LSF_ENVDIR/lsf.shared` file to enable the docker container configuration.

```
Begin Resource
RESOURCENAME  TYPE    INTERVAL INCREASING  DESCRIPTION        # Keywords
…
   docker     Boolean ()       ()          (Docker container)
…
End Resource
```

- Update `lsf.cluster.<boa_appliance>`:
  This is a cluster configuration file and all LSF hosts are listed in this file. Ensure docker is configured under the RESOURCES column for MASTER LSF host in the `$LSF_ENVDIR/lsf.cluster.boa_appliance` file.

```
Begin   Host
HOSTNAME  model    type        server  RESOURCES    #Keywords
…
<master lsf hostname>   !      !            1     (mg docker)
…
End     Host
```

- Update `lsb.applications`:
  The `lsb.applications` file defines application profiles. The CONTAINER parameter in the application profile enables LSF to use a Docker container for jobs that are submitted to the application profile. The CONTAINER parameter shall be specified with Bayesian optimization job image name, environment variable, docker network and labels. This configuration is optional at the time of LSF Installation. In case admin is not aware of required configuration this step can be skipped however admin must configure lsb.applications file post BOA configuration.

```
Begin Application
NAME = boa
DESCRIPTION = Execute application through docker containers
CONTAINER = docker[image(boa/bayesian-opt-job:<build>-<ppc64le>) options( --rm -e
MONGO_HOST=boa-mongo -e MONGO_PORT=27017 -e MONGO_USER=mongoadmin -e MONGO_PASS=secret -e
MONGO_DB_NAME=boa -e CELERY_BROKER_URL=redis://boa-redis:6379 -e
CELERY_RESULT_BACKEND_URL=redis://boa-redis:6379 -e MQTT_BROKER_HOST=boa-mosquitto -e
MQTT_BROKER_PORT=1883 -e MQTT_BROKER_WS_PORT=9001 -e MQTT_BROKER_HOST_EXTERNAL=< hostip> -e
MQTT_BROKER_PORT_EXTERAL=<nginx.http.port> -e MQTT_BROKER_WS_PORT_EXTERNAL=< nginx.http.port> -
e MQTT_BROKER_WS_URL=ws://<hostip>:< nginx.http.port>/mqttws/ -v /usr/lib64:/usr/host/lib64 --
network=<use build id by replacing '.' by '-'>_boa-network --label com.ibm.boa.experiment="1")]
End Application
```

- Start cluster:
  Run the lsfstartup command to start the cluster:

```
lsfstartup
Type “yes” and Enter password of host machine.
```

- Verify LSF installation:
  Run the `lsid` command to verify LSF installation. An output similar to the following screen is displayed:

```
IBM Spectrum LSF Standard 10.1.0.0, Apr 10 2020
```

- Install LSF fix pack 9:

  LSF fixpack is required to have latest features and fixes installed. In BOA appliance it is required to support docker container jobs and GPU related features and fixes.

  Copy LSF fixpack file:

```
Go to the patch install directory: cd $LSF_ENVDIR/../10.1/install/
Copy the patch file to the install directory $LSF_ENVDIR/../10.1/install/
Run
```

```
badmin hclose all
badmin qinact all
```

- Run patchinstall:
  Run the `patchinstall` command to install the patch:

```
./patchinstall <patch file>
Type “yes” for User Input
```

- After patch install:
  After patch install restart all services after the installation of patch:

```
badmin hshutdown all
lsadmin resshutdown all
lsadmin limshutdown all

lsadmin limstartup all
lsadmin resstartup all
badmin hstartup all

badmin hopen all
badmin qact all
```

- Verify LSF fixpack installation:
  Run the `lsid` command to verify LSF installation. An output similar to the following screen is displayed:

```
IBM Spectrum LSF Standard 10.1.0.9, Apr 10 2020
```

LSF logs:

- You can view LSF logs in the `/home/share/lsf/log` directory in the host system. The LSF logs can be configured in the `install.config` file.

Installing BOA

The BOA installation program is used to perform a one-time setup of the BOA Docker images and to create a small installation directory of the supporting configuration files. The BOA installation program is a simple, silent, and response file-based installer. You need to configure a few installation properties during the BOA installation process. The BOA installation program also helps you to uninstall the BOA.

Unpacking the BOA software package:
Unpack the BOA software distribution archive. BOA is delivered as a compressed archive file in the tgz format. Create a temporary directory for this package and extract the BOA installation program files from the archive.

Preparing the installation silent response file:
The BOA installation program is a non-interactive silent installer. The user input is in a properties file that the BOA installation program reads. You need to update the properties in this file by editing the sample installation property at `BOA_TEMP_DIR/installer/samples/boa_installer.properties`.

There are two properties in the `boa_installer.properties` sample file.

Table 1. Property key in the BOA installer file

Property key	Description	Usage
LICENSE_ACCEPT	This key specifies whether the user accepts the BOA license agreement.	Allowable values are yes and no. BOA is installed if the value yes is specified. Specification of yes constitutes acceptance of BOA license.

Table 1. Property key in the BOA installer file (continued)		
Property key	**Description**	**Usage**
BOA_INSTALL_DIR	This key specifies the installation directory in which BOA is installed.	Specify an absolute directory path that the user running the BOA installation program has permission to create. You must not give an existing directory name here.

Running the BOA installation program:
After you prepared the silent response file, you can start installation. Run the `BOA_TEMP_DIR/boa_installer.sh shell` script to start the installation.

```
cd /boabuildtemp/installer/bin
./boa_installer.sh -a install -r /boabuildtemp/installer/samples/boa_installer.properties
```

The BOA installation program supports the following syntax:

```
Usage:
 boa_installer.sh -h                                    Display this help message.
 boa_installer.sh -a install -r <response_file>      Install BOA with properties defined in
<response_file>.
 boa_installer.sh -a uninstall -r <response_file>    Uninstall BOA silently with properties
defined in <response_file>.
```

BOA installation program supports the following options:

Table 2. Options in the BOA installation program		
Option	**Description**	**Usage**
`-h`	Displays a short usage for the installer	
`-a`	Specifies the action that the installer performs	Specifies install or uninstall the BOA
`-r`	Path to the silent response file used for installing or uninstalling BOA	Specifies the path to the `boa_installer.properties` file

During the installation process, the installer runs the following process:

- Validate the arguments
- Validate that the response file exists and contains the mandatory two properties and that LICENSE_ACCEPT is true.
- Validate that no file or directory exists at the location that is specified by BOA_INSTALL_DIR.
- Validate that Docker engine is installed, and the Docker daemon is running.
- Create the BOA installation directory.
- Copy the BOA files from the temporary directory to the BOA installation directory.
- Run the command-line interface (CLI) to load the image in the local Docker registry.
- Validate that the docker-compose is already installed, and deploy docker-compose if it is not currently installed. If user executes `boa_installer.sh` file without accepting the license in `boa_installer.properties` file then accept the license and remove the `BOA_INSTALL_DIR`.

Troubleshooting installation problems

Specific messages are displayed if the BOA installation program is unable to complete the installation process. The following are common conditions and corrective responses to take before retrying.

Table 3. Troubleshooting BOA installation

Message	Workaround
Either the `-a` or `-r` options are not specified.	All invocation of the BOA installation program require the action and response file. Add the missing option and retry the installation.
A value is not provided for the `-a` or `-r` options.	Specify one of the allowable actions with the `-a` option and specify the response file that contains installation properties with the `-r` option.
The response file does not exist.	Specify the path to an existing response file by using the `-r` option.
The LICENSE_ACCEPT or BOA_INSTALL_DIR properties are not defined or commented out in the response file.	Modify the response file to ensure that the specified properties are defined.
The LICENSE_ACCEPT value is not `yes`.	Modify the response file to ensure that the value `yes` is specified for the LICENSE_ACCEPT property.
The BOA_INSTALL_DIR value points to an existing file or directory.	Ensure that the value you specify for BOA_INSTALL_DIR points to a non-existent directory on the BOA host.
The Docker engine is not installed.	Use the instructions in the Host Prerequisite section to install the Docker engine.
The Docker engine is installed, but the Docker daemon is not running.	Start the Docker daemon with `systemctl start docker` and verify that the daemon is successfully started with `systemctl status docker`.
The BOA_INSTALL_DIR is not able to be created.	Ensure that the permissions to create the BOA_INSTALL_DIR are compatible with user who runs the BOA installation program.
Errors occurred in copying files from the temporary installation directory to the BOA_INSTALL_DIR.	
Errors occurred in loading the BOA images into the local Docker registry.	Check the detailed error messages. Ensure that the storage area used by the Docker daemon has adequate free space to accommodate the 15 GB of storage for the BOA Docker images. The default path for this storage area is `/var/docker/lib`.

Here is an camelback example:

```
"""
Licensed Materials - Property of IBM
“Restricted Materials of IBM”

5765-R17

© Copyright IBM Corp. 2020  All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp
"""

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import os, sys
import time
from common.boa_examples_utils import BoaExamplesUtils
from boaas_sdk import BOaaSClient

example_description="""
        This example demonstrates basic BOA usage for a simple optimization problem.

        The BOA SDK has been designed to be simple to use, but flexible in the range of
        configurations available for tailoring the optimization. This is achieved using
        the BOaaSClient object, which facilitates all communication with the BOA server.
        The optimization configuration is handled via a Python dictionary (or
        equivalently a JSON file).

        The camelback example is a two dimensional optimization problem that optimizes
        the six humped camelback function. This function has global minima
        f(x) = -1.0316 at x = (0.0898, -0.7126) and (-0.0898, 0.7126))
        """

"""
Perform parsing of common commandline arguments shared by all BOA examples.
"""
args = BoaExamplesUtils.parse_commandline_args(example_description, default_epochs=40)
hostname = BoaExamplesUtils.get_connection_url(args)
print("Connecting to host: {}".format(hostname))
boaas = BOaaSClient(host=hostname, ca_cert_path=args.ca_cert_path)

def camelback6(x):
    # Six-hump camelback function
    x1 = x[0]
    x2 = x[1]
    f = (4 - 2.1 * (x1 * x1) + (x1 * x1 * x1 * x1) / 3.0) * (x1 * x1) + x1 * x2 + (-4 + 4 * (x2 *
x2)) * (x2 * x2)
    return f


experiment_config = {
    "name": "Reg camelback function",
    "domain": [
        {
            "name": "x1",
            "min": -2,
            "max": 2,
            "step": 0.01
        }, {
            "name": "x2",
            "min": -1,
            "max": 1,
            "step": 0.01
        }
    ],
    "model": {"gaussian_process": {
        "kernel_func": "Matern52",
        "scale_y": True,
        "scale_x": False,
        "noise_kernel": True,
        "use_scikit": False
    }},
    "optimization_type": "min",
    "initialization": {
        "type": "random",
        "random": {
            "no_samples": 3,
            "seed": None
        }
    },
    "sampling_function": {
        "type": "expected_improvement",
        "epsilon": 0.03,
        "optimize_acq": False,
        "outlier": False,
```

Uninstalling BOA

The following topic describes how to uninstall IBM® Bayesian Optimization Accelerator.

Uninstallation of BOA is performed using the BOA Installation Program and specifying an action of “uninstall” and a reference to the silent response file for the BOA installation. This should be the same silent response file that you modified when performing the original installation.

During uninstallation, the BOA Docker images previously loaded into the local Docker registry during installation are removed from the local Docker registry. The BOA installation directory is not removed when performing an uninstallation using the BOA Installation Program. You are responsible for deleting this directory if you no longer have a need for the configuration files and log files contained in the installation directory.

To perform an uninstallation of BOA, run the BOA_TEMP_DIR/boa_installer.sh shell script specifying the uninstall action and a path to the silent response file.

```
cd /boabuildtemp/installer/bin
./boa_installer.sh -a uninstall -r /boabuildtemp/installer/samples/boa_installer.properties
```

Initial configuration of BOA

After installing BOA, complete the initial configuration of BOA. The configuration process produces a set of configuration files, called a deployment, that can be used to run BOA.

The BOA configuration tool configures BOA. You can customize parameters during the configuration process. For an initial configuration of BOA, you can take the default configuration parameters.

- In the LSF configuration section, update the `/opt/IBM/boa/installer/samples/boa_config.yml` file for `lsfadmin` user's password. The following example displays an example of updating password for the `lsfadmin` user.

```
lsf:
  user: lsfadmin
  password:
```

- To run the BOA configuration tool with the default parameters, use the following commands:

```
cd  /opt/IBM/boa/installer/bin
./boa_config_tool.sh -a configure -c /opt/IBM/boa/installer/samples/boa_config.yml
```

  The **`configure`** command creates a set of deployment files under the directory `/opt/IBM/boa/deployments/default` using the default configuration parameters.

- Copy the `lsb.applications` file generated by the boa configuration tool to the LSF configuration directory if you have configured the `lsb.applications` file during the LSF installation.

  Run the following command to copy the file:

```
cp /opt/IBM/boa/deployments/default/config/lsf/lsb.applications $LSF_ENVDIR/lsbatch/
boa_appliance/configdir/lsb.applications
```

  Type `yes` to overwrite the file.

  Run the `badmin reconfig` command to reconfigure LSF:

```
badmin reconfig
```

  An output similar to the following screen is displayed on the successful execution of the command:

```
Checking configuration files ...
No errors found.
Reconfiguration initiated
```

Directory structure of the BOA deployment installation

After unpacking the BOA installation package, the following directory structure is created. In the table, BOA_HOME represents the root of the BOA installation directory.

Table 4. Directory structure and content of the directory

Directory	Content
BOA_HOME/images	The `.tar` file of the Docker images for BOA. The `–action load_images` of the BOA configuration tool is used to load these images into the local Docker registry.
BOA_HOME/installer/samples	This directory contains sample BOA configuration files. You can use these files as starting points when creating your BOA deployments.
BOA_HOME/installer/bin	This directory contains the BOA configuration tool and other scripts for managing your BOA deployments.
BOA_HOME/installer/templates	This directory contains templates for the configuration files that are associated with a BOA deployment. The BOA configuration tool combines configuration information from the BOA configuration file with these files to produce a BOA deployment.
BOA_HOME/deployments	This directory is the root directory for generated BOA deployments that are created by the BOA configuration tool.
BOA_HOME/deployments/deployment_name	This is the directory in which the deployment files for a BOA deployment that is named `deployment_name` is stored.
BOA_HOME/sdk	This directory contains an installable package for the BOA Python SDK.

Starting BOA deployment services

Once you have generated a set of deployment files with the BOA configuration tool, start the BOA services. As BOA is a Docker-based application, use the docker-compose program to manage the BOA services.

- To start the BOA services, run the following commands:

```
cd  /opt/IBM/boa/deployments/default
docker-compose up -d
```

The **`docker-compose up`** command create a running BOA deployment . After a successful execution of `docker-compose up -d`, all the BOA microservices are started, and ready to accept requests from the BOA SDK client programs and users. For more information, see “Configuration and deployment concepts” on page 15.

URLs for BOA Services

After starting the BOA services using docker-compose, BOA URLs for those services are available. The actual URLs are constructed based on the configuration parameters you specified when running the configure action of the BOA Configuration Tool.

The following configuration elements contribute to the construction of the URLs:

Table 5. Elements of BOA URL	
Configuration elements	**URL contribution**
boa_appliance.fqdn	Is used as hostname portion of all URLs.
boa_appliance.ip_address	Is used as the hostname for IP address-based URLs.
nginx.http.port	Is used as the port for all HTTP-based URLs except for the BOA Experiment Viewer URL.
nginx.https.port	Is used as the port for all HTTPS-based URLs except for the BOA Experiment Viewer URL.

FQDN-based URLs:
Using the above configuration elements, the following fully-qualified domain named-based URLs are available after the startup of the BOA services.

Table 6. URL format	
URL	**URL format**
BOA Server API URL - HTTP	http://boa_appliance.fqdn:nginx.http.port/api
BOA Server API URL - HTTPS	https://boa_appliance.fqdn:nginx.https.port/api
BOA Admin Server API URL - HTTPS	http://boa_appliance.fqdn:nginx.http.port/api/admin
BOA Admin Server API URL - HTTPS	https://boa_appliance.fqdn:nginx.https.port/api/admin
BOA Admin UI URL - HTTP	http://boa_appliance.fqdn:nginx.http.port
BOA Admin UI URL - HTTPS	https://boa_appliance.fqdn:nginx.https.port
BOA Experiment Viewer - HTTPS	https://boa_appliance.fqdn:nginx.https.port/exp_viewer

IP address-based URLs: The IP address-based forms of the URLs are derived by substituing the value of the boa_appliance.ip_address configuration element in place of the boa_appliance.fqdn configuration element in the above table. Example URLs using Default Ports The following table lists available URLs assuming:

- Default port configuration
- A FQDN of myserver.boa.com
- An IP address of 172.20.181.33

Table 7. URL format of IP address-based URLs	
URL	**URL format**
BOA Server API URL - HTTP	http://myserver.boa.com:80/api
	http://172.20.181.33:80/api
BOA Server API URL - HTTPS	https://myserver.boa.com:443/api
	https://172.20.181.33:443/api
BOA Admin Server API URL - HTTPS	http://myserver.boa.com:80/api/admin
	http://172.20.181.33:80/api/admin

Table 7. URL format of IP address-based URLs (continued)	
URL	**URL format**
BOA Admin Server API URL - HTTPS	https://myserver.boa.com:443/api/admin
	https://172.20.181.33:443/api/admin
BOA Admin UI URL - HTTP	http://myserver.boa.com:80
	https://172.20.181.33:443
BOA Admin UI URL - HTTPS	https://myserver.boa.com:443
	https://172.20.181.33:443
BOA Experiment Viewer - HTTPS	https://myserver.boa.com:443/exp_viewer
	https://172.20.181.33:443/exp_viewer

Managing your BOA deployment with the docker-compose program:
The BOA runtime consists of a set of microservices running as Docker containers. The containers are wired together and operationally managed using orchestration software called docker-compose. You can use this program to start and stop the BOA services, to view logs from the services, and to perform other operational tasks such as querying BOA service status.

The capabilities provided by docker-compose are described in the online help displayed when you specify "docker-compose" from a terminal on the BOA Appliance. You can also view the command reference for the docker-compose program at https://docs.docker.com/compose/reference/.

For BOA, you must always run the docker-compose CLI from the directory that contains your BOA deployment files. This directory is located at: `BOA_INSTALL_DIR/deployments/<deployment-name>` where

- BOA_INSTALL_DIR is the BOA installation directory you specified when you ran the BOA Installation Program.
- <deployment-name> is the name you specified for the –deployment option when configuring your BOA deployment using the configure action of the BOA Configuration Tool. If you didn't specify a named deployment, a deployment name of "default" is used and the following deployment directory is created to contain your BOA deployment: BOA_INSTALL_DIR/deployments/default

Always change to your named deployment directory and run the docker-compose CLI from that directory location.

Setting up your SDK client environment:
After installation and configuration of BOA, you can develop programs using the BOA Python SDK to invoke the BOA Server APIs to create and run experiments. You will need to setup a Python 3.6-based system where your BOA SDK programs run.

This system must include:

- Python 3.6
- All Python package dependencies required by the BOA Python SDK
- The BOA Python SDK package
- Any additional Python package dependencies that your experiment Python code requires

The BOA Python SDK requires the following Python packages and versions to be installed:

- numpy>=1.12.0
- scikit-learn>=0.18
- paho-mqtt==1.3.1
- requests==2.18.4
- scipy>=0.17.0

- pandas>=0.22.0
- jupyter>=1.0.0

You use the Python pip utility to install the BOA Python SDK in your python environment. Follow these general steps for the install.

- Copy the BOA Python SDK archive found under BOA_INSTALL_DIR/sdk/lib/boaas_sdk-1.1.0-py3-none-any.whl on the BOA Appliance to the system where you will run your BOA SDK program.
- If you are using a packaging system that supports a virtual environment concept (like virtualenv or Anaconda), you should activate the virtual environment you are using for your BOA SDK program development.
- Install the BOA Python SDK

```
pip install  boaas_sdk-1.1.0-py3-none-any.whl
```

Requirements for running BOA SDK programs:
The system you are running your BOA SDK program on needs to satisfy the requirements specified in this section.

- **Port Access**: If you are using the HTTP-based URL for the BOA Server API, you need to have network access from the system where your BOA SDK program is executing to the port on the BOA Appliance specified by configuration element nginx.http.port in your BOA deployment. The default number for this port is port 80. If you are using the HTTPS-based URL for the BOA Server API, you need to have network access from the system where your BOA SDK program is executing to the port on the BOA Appliance specified by configuration element nginx.https.port in your BOA deployment. The default number for this port is port 443.
- **SSL/TLS CA Certificate Setup**: If you are using the HTTPS-based URL for the BOA Server API, you need to transfer the CA Certificate for you deployment from the BOA Appliance to the system where your BOA SDK program is executing.

On the BOA Appliance, this file is found in path BOA_INSTALL_DIR/deployments/<deployment-name>/config/ssl/ca.crt where BOA_INSTALL_DIR is the root of your BOA installation directory and <deployment-name> is the name of your BOA deployment. The default <deployment-name> is “default”.

Once you have transferred the Certificate, you need to add it to the set of trusted CA certificates in your operating system. This varies based on the operating system you are using.

Configuration and deployment concepts

Configuration is the process of parsing a set of customer-provided configuration parameters and creating a working BOA deployment from those parameters. Configuration in BOA is carried out by the BOA Configuration Tool.

The input to the BOA Configuration Tool is a YAML file that is called the `BOA Configuration File`. This file provides parameters for all customization that a customer can define to BOA.

A BOA deployment is the complete set of configuration files that are created by the BOA configuration tool from the input that is provided by the user in the BOA configuration file. Examples of some of the files that are generated for a BOA deployment from the BOA configuration tool are:

- **`docker-compose.yml`**– This is the main orchestration file for BOA. It defines the services that compose a deployment, their associated Docker images, and the wiring between the services. Wiring represents the specific environment variables, connection properties, and configuration files that are used to orchestrate the execution of the services that comprise a BOA deployment.
- **`SSL/TLS Files`** – These are certificates and private key files that are used by BOA to implement transport encryption. These can be generated by the BOA Configuration Tool or can be supplied by the customer.
- **`nginx.conf`** – This is the configuration file for the NGINX container that is run for BOA . This component provides a reverse proxy and web server function for BOA.

- **`lsfjob.cfg`** – This is the configuration file for LSF jobs. Boa admin can use this file to specify application profile for the job, Memory requirement for the job.
- **`lsb.applications`** - This file is LSF configuration file that defines the docker container application in BOA which is having configuration for boa optimization job to run like boa optimization image name, environment variables, network configuration, labels. Config tool generates this file so admin can use this file and replace LSF config file at $LSF_ENVDIR/lsbatch/boa_appliance/configdir/lsb.applications by BOA_INSTALL_DIR/deployments/<deploymentname>/ config/lsf/lsb.applications file created by config tool. Admin must run "badmin reconfig" when reconfiguring this file.

Once a BOA deployment is created, the user has two primary tools to operationally manage the BOA deployment:

- **Docker-compose program**: The docker-compose program provides a command-line interface (CLI) to operationally administer the BOA deployment. The docker-compose program uses the service definitions in the `docker-compose.yml` deployment file to support the operational activities like starting and stopping the BOA deployment, or managing log files generated by the BOA services.
- **BOA Administrative UI**: The BOA Administrative UI is a graphical user interface for displaying information about the BOA deployment, and managing the deployment of BOA.

The BOA configuration tool can be used to generate multiple named BOA deployments. For example, you might generate a named BOA deployment for your existing production deployment and a separate BOA deployment to test upgrades and fix packs.

The BOA configuration tool can also be used when performing the initial setup of a BOA deployment or it can be used to modify an existing BOA deployment that you previously created.

So, in summary:

- For an initial setup of BOA, the customer customizes the BOA deployment by first defining their wanted configuration in a BOA Configuration File.
- The customer runs the BOA configuration tool to parse and validate the BOA configuration file and generates a set of configuration files that are called a BOA deployment.
- After generating the BOA deployment, the customer uses the docker-compose program and the BOA Administrative UI to manage the BOA deployment files and operationally administer the deployment.
- The BOA Configuration Tool can also be used to modify an existing deployment and to set up multiple BOA deployments with different names.

BOA configuration tool reference

Running the BOA configuration tool

The BOA configuration tool is located in file `boa_config_tool.sh` in the `BOA_HOME/installer/bin` directory. BOA configuration tool is a command-line interface (CLI) that accepts a set of arguments from the user.

Running the BOA configuration tool

The BOA configuration tool is located in the `boa_config_tool.sh` file, in the `BOA_HOME/installer/bin` directory. The BOA configuration tool is a command-line interface (CLI) that accepts a set of arguments from the user.

You should always run the BOA configuration tool from the `BOA_HOME/installer/bin` directory. If you attempt to run the BOA configuration tool from another directory, an error message is displayed, and the command fails. The messages generated by the BOA configuration tool are written to standard output of the terminal.

The BOA configuration tool returns an exit code that indicates the success of the command execution.

- An exit code of `0` indicates that the command was successful.

- An exit code of 1 indicates that the command failed. In this case, additional error messages identify the nature of the failure.

1. When running the BOA configuration tool, change to the directory containing the `boa_config_tool.sh` shell script.
2. Run the `boa_config_tool.sh` script passing arguments identifying the operations and parameters for the command.

   The BOA configuration tools accepts arguments in the following format:

   - Arguments have both a short form and a long form.
   - In the short form, the argument is specified with a dash character followed by a single character uniquely identifying the argument. For example, `-a` is the short form used to specify the action that the BOA configuration tool should perform.
   - In the long form, the argument is specified with two dash characters followed by multiple characters uniquely identifying the argument. For example, `--action` is the long form used to specify the action that the BOA configuration tool should perform.
   - You can freely mix short and long forms of the arguments when running the BOA configuration tool.
   - Some arguments also have argument values. For those arguments, the argument value is provided after the short or long form of the argument and a space. For example, in `--action configure`, the argument value is configure for the `–action` argument.

   Supported actions:
   All invocations of the BOA configuration tool include the specification of the action to perform. The action is specified with either the `-a` short form or the `–action` long form.

Table 8. Supported actions

Action	Description
`configure`	This action validates the configuration parameters in a BOA configuration file and generates a BOA deployment for that configuration.
`validate`	This action validates the configuration parameters in a BOA configuration file without generating BOA deployment files.

   Options supported for each action:
   In addition to the action that is required for each invocation of the BOA configuration tool, some actions have additional optional arguments that may be specified. These optional arguments are scoped to the action. Not all optional arguments apply to each action. The following table specifies the optional arguments, the actions with which they apply, any default value, and a description of the argument.

Table 9. Options supported for each action

Argument Short Form	Argument Long Form	Action(s)	Default	Description
`-c`	`--configfile`	configure, validate	`BOA_HOME/installer/samples/boa_config.yml`	This argument identifies the BOA configuration file to be processed. The BOA configuration file identifies the aspects of a BOA deployment that can be customized by the customer. This is a YAML file.
`-d`	`--deployment`	configure, ivt	default	This argument identifies the named BOA deployment that is either generated (configure action) or verified (ivt action).
`-o`	`--overwrite`	configure	N/A	This argument instructs the BOA configuration tool that it is acceptable to overwrite the contents of an existing BOA deployment directory.
`-n`	`--no-overwrite`	configure	N/A	This argument instructs the BOA configuration tool that the contents of an existing BOA deployment directory should not be overwritten.
`-f`	`--force`	configure	N/A	This argument instructs the BOA Configuration Tool to bypass any failed validation checks and to generate deployment files even if one or more checks fails.

Table 9. Options supported for each action (continued)

Argument Short Form	Argument Long Form	Action(s)	Default	Description
-nf	--no-force	configure	N/A	This argument instructs the BOA Configuration Tool to not generate deployment files unless all validation checks are satisfied.
-h	--help		N/A	This argument displays usage and help information about the BOA configuration tool.

Summary of BOA Configuration Tool Syntax:

```
usage: boa_config_tool.sh [-h] --action {configure,validate}
                          [--configfile CONFIGFILE] [--deployment DEPLOYMENT]
                          [--overwrite] [--no-overwrite]
                          [--force] [--no-force]

optional arguments:
  -h, --help            show this help message and exit
  --action {configure,validate}, -a {configure, validate}
                        configuration action to perform (default: None)
  --configfile CONFIGFILE, -c CONFIGFILE
                        path to configuration input YAML file used to
                        configure BOA (default: /boa_install_root_dir/installe
                        r/samples/boa_config.yml)
  --deployment DEPLOYMENT, -d DEPLOYMENT
                        name of BOA deployment to be configured (default:
                        default)
  --overwrite, -o       overwrite existing deployment files (default: False)
  --no-overwrite, -n    do NOT overwrite existing deployment files (default:
                        False)
```

Action Details:

- `configure` action:

  The configure action accepts as input the BOA configuration file. The file is parsed and validated. If all validation checks are satisfied, the action generates a BOA deployment (set of runnable configuration files).

  The configure action can be used to generate an initial deployment of BOA or can be used to update an existing deployment.

  The `–configfile` argument is used to identify the BOA configuration file to be used for the configuration process. The `–deployment` option can be used to specify the named BOA deployment that is generated by the configure action. The `–overwrite` and –no-overwrite options can be used to control whether the configure action will overwrite or preserve a previously generated BOA deployment.

Table 10. Description of the configure action	
Command	**Description**
`./boa_config_tool.sh --action configure`	Reads and validates a BOA configuration file from the default location (`BOA_HOME/installer/sample/boa_config.yml`). If the validation is successful, generates a runnable BOA deployment under `BOA_HOME/deployments/default` directory.
`./boa_config_tool.sh--action configure --configfile /tmp/qa_config.yml --deployment QA --no-overwrite`	Read and validate the BOA configuration file located under `/tmp/qa_config.yml`. If the validation is successful, generate a runnable BOA deployment under `BOA_HOME/deployments/QA`. If deployment files already exist under `BOA_HOME/deployments/QA`, the files are not overwritten.

- `validate` action:

  The validate action performs the validation checks, that are performed as part of the configure action without generating the BOA deployment files. This action can be used to test out changes to the BOA configuration file you are developing or to verify that the BOA Appliance Host prereq software is deployed and configured correctly.

Table 11. Description of the validation action	
Command	**Description**
`./boa_config_tool.sh --action validate`	Reads and validates a BOA configuration file from the default location (`BOA_HOME/installer/sample/boa_config.yml`).
`./boa_config_tool.sh --action validate --configfile /tmp/qa_config.yml`	Read and validate the BOA configuration file located under `/tmp/qa_config.yml`.

BOA configuration file reference

The BOA configuration file is a YAML file that contains your configurations and customizations for the BAO environment. When you run the configure or validate actions of BOA configuration tool, you can provide an argument `–configfile` that references the path to the BOA configuration file to be used.

A sample BOA configuration file is in the BOA installation tree under the `BOA_HOME/installer/samples/boa_config.yml` file. If you did not specify the `-configfile` argument, the BOA configuration tool uses this file.

Validation of the configuration

The file must be well-formed YAML. The BOA configuration tool parses the BOA configuration file and exits immediately if any parsing errors or syntax errors are detected in the BOA configuration file.

In addition to validating that the BOA configuration file is well-formed YAML, the BOA configuration file also performs additional validation checks against the configuration values you provide in the BOA configuration file.

Examples of these checks include:

- Validating that the elements representing ports are syntactically valid port numbers
- Validating that the elements representing IP addresses are syntactically valid Ipv4 or Ipv6 addresses

Required and optional configuration elements

Unless otherwise noted, values must be provided for all configuration elements. An exception to this rule are the configuration for Spectrum LSF configuration elements. If you configure BOA Appliance to use Python Celery as the job manager, then the Spectrum LSF configuration elements need not be defined.

Some configuration elements accept the value `default` to indicate that BOA configuration tool should derive the value for that configuration element.

Basic configuration elements for BOA

The basic networking configuration for the BOA Appliance is described in the initial set of the configuration elements in the BOA configuration file. These are described in this section.

Sample configuration

The following text box shows a sample of defining the basic configuration elements for BOA.

```
version: '1.0'
boa_appliance:
  fqdn: default
  ip_address: default
  docker_subnet: "172.178.0.0/16"
  ssh_port: 22
```

Element descriptions

The following table describes the basic configuration elements.

Table 12. Configuration elements description

Configuration elements	Description	Format	Validation
`version`	Defines the version of the BOA configuration file	Must be 1.0	Validates that one of the allowable values is specified
`boa_appliance`	Nests various elements related to the BOA Appliance	N/A	N/A
`boa_appliance .fqdn`	The fully qualified domain name for the BOA Appliance	`default` or a fully-qualified domain name	Validates that either `default` is specified or a valid well-formed fully qualified domain name (FQDN) is specified
`boa_appliance .ip_address`	The primary IP address for the BOA Appliance	`default` or an AP address	Validates that either `default` is specified or a valid Ipv4 or Ipv6 Internet Protocol address is specified
boa_appliance .docker_subnet	The IP network specification for the Docker bridge network that docker-compose creates for this BOA deployment	IP network specification with valid mask	Validates that the value is a valid Docker subnet specification

Table 12. Configuration elements description (continued)			
Configuration elements	**Description**	**Format**	**Validation**
boa_appliance .ssh_port	The port that the SSH daemon listens to for the BOA Appliance	port number	Validates the specified value is an integer in the valid range of port numbers

Note:

- If default is specified for `boa_appliance.fqdn` or `boa_appliance.ip_address`, then the BOA configuration tool will derive those values by invoking standard Linux services.
- The sample value for `boa_appliance.docker_subnet` normally works. If you want to have multiple BOA deployments actively running on the same BOA Appliance, you need to ensure that all specified Docker bridge networks are not overlapping. The `boa_appliance.docker_subnet` is used to configure such an environment.
- The SSH daemon must be running on the BOA Appliance, also the BOA microservices must be running in the Docker containers.

Configuring a job manager for BOA

The BOA Appliance spawns BOA jobs that perform the Bayesian optimization tasks. IBM Spectrum LSF is embedded inside the BOA Appliance and is the default job manager. You can also configure the BOA Appliance to use Python Celery as the BOA job manager. This section details how that is configured.

Sample configuration

The following text box shows a sample of configuring a job manager for BOA.

```
job_manager: lsf
```

Element description

The following table describes the job manager-related configuration elements.

Table 13. Description of the job-manager configuration elements			
Configuration elements	**Description**	**Format**	**Validation**
`job_manager`	Specifies the job manager used by BOA Appliance when spawning Bayesian optimization jobs	Must be one of either lsf or celery	Validates that one of the allowable values is specified

Note:

- IBM Spectrum LSF provides advanced load sharing and job scheduling features. It is the default job scheduler.
- Python Celery can be used if setting up BOA deployments in a unit or integration test environment doesn't have LSF installed.
- The configuration for Python Celery uses a single Celery worker and up to *n* number of concurrent Celery subprocesses that are managed by the Celery worker – here *n* represents the number of installed CPU cores. This configuration is not currently customizable using the BOA configuration tool.

Configuring BOA and LSF integration

IBM Spectrum LSF is embedded inside the BOA Appliance and is the default job manager. This section details the configuration elements that can be customized when integrating the BOA Appliance with LSF. These elements must be provided and are validated when you specify the `job_manager` configuration element as `lsf`.

There are two partition of configuration elements related to LSF:

- Configuration elements that specify how BOA services that are running in Docker communicate with the LSF services for scheduling and managing BOA optimization jobs. These elements include the specification of what user credentials are used when calling LSF and the names of the cluster file and profile files in your LSF deployment.
- Configuration elements that identify the resource requirements that BOA sends to LSF when scheduling BOA optimization jobs. These configuration elements are used to derive the memory and GPU requirements sent to LSF when a BOA optimization job is submitted to LSF.

Sample configuration

The following text box shows a sample of configuring LSF Integration with BOA.

```
lsf:
  user: lsf_user
  password: somepassword
  cluster_file: /home/share/lsf/conf/lsf.cluster.boa_appliance
  profile_file: /home/share/lsf/conf/profile.lsf
```

Elements description

The following table describes the job manager LSF-related configuration elements.

Table 14. Description of the BOA-LSF integration elements

Configuration elements	Description	Format	Validation
lsf	Nests various elements related to LSF	N/A	N/A
lsf.user	The LSF user under which BOA interactions with LSF are performed	Linux user name	Validates that the user name is specified correctly, and an SSH connection to the BOA Appliance host can be established with that user
lsf.password	The password associated with lsf.user	Linux password	Validates that the user name is specified correctly, and an SSH connection to the BOA Appliance host can be established with that user
lsf.cluster_file	LSF cluster file for the BOA LSF deployment	Absolute file path to the LSF cluster definition file on the BOA Appliance host	Validates that the file exists and can be accessed by the user specified in `lsf.user`

Table 14. Description of the BOA-LSF integration elements (continued)			
Configuration elements	**Description**	**Format**	**Validation**
lsf.profile_file	LSF profile file for the BOA LSF deployment	Absolute file path to the LSF profile file on the BOA Appliance host	Validates that the file exists and can be accessed by the user specified in `lsf.user`
lsf.job_reqts	Nest a collection of configuration elements that govern how resource requirements for BOA jobs are calculated	N/A	N/A
lsf.job_reqts .mem_safety_factor	Service memory in bytes	Non-negative integer number of bytes	Validates that the specified value is a non-negative integer value
lsf.job_reqts .mem_per_grid_in_bytes	Memory per grid in bytes	Non-negative integer number of bytes	Validates that the specified value is a non-negative integer value
lsf.job_reqts .gpus_per_job	Number of GPUs to reserve for each BOA optimization job	Non-negative integer number of GPUs	Validates that the specified value is a non-negative integer value

Note:

- BOA microservices run as Docker containers inside a Docker bridge network.
- IBM Spectrum LSF is installed on the BOA Appliance host in a single node LSF master configuration.
- When BOA Docker microservices need to perform LSF actions on the BOA Appliance host, they establish an SSH connection to the BOA Appliance host. The `lsf.user` and `lsf.password` configuration elements are used as the credentials for the connection.
- If such a connection can be established, it is used to validate that the user named in the `lsf.user` element can access the host files referenced in the `lsf.cluster_file` and `lsf.profile_file` elements.
- If the above validations are successful, the BOA configuration tool finally attempts to source the profile identified in `lsf.profile_file` and run the `lsid` command on the host.
- If all above validations are successful, the LSF and BOA integration is in a working state.
- The SSH daemon must be configured correctly and running, and the LSF daemons must be started when the BOA configuration tool is invoked.

LSF job configuration

This is the configuration file that BOA appliance requires to enable docker container LSF jobs and BOA admin can use this file to specify application profile for the job, Memory requirement for the job, environment variable for the job.

The following text box shows a sample of configuring LSF job.

```
LSF_JOB_TYPE="docker"
LSF_APP_NAME="boa"
LSF_JOB_ENV="all"
LSF_JOB_DEFAULT_MEM_REQ=50
```

The following table describes the LSF job related configuration elements:

Table 15. Description of the LSF job elements			
Configuration elements	**Description**	**Format**	**Default**
LSF_JOB_TYPE	Type of the job	String	LSF_JOB _TYPE="docker"
LSF_APP_NAME	Name of the application defined in lsb.applications	String	LSF_APP_NAME="boa"
LSF_JOB_ENV	Environment variable for the LSF job	all, var1=value1, var2=value2	LSF_JOB_ENV="all"
LSF_JOB_DEFAULT _MEM_REQ	Percentage of physical memory required for each LSF job	Number	LSF_JOB_DEFAULT _MEM_REQ=50

Elements description

Table 16. Elements description of the LSF job configuration			
Configuration elements	**Description**	**Format**	**Default**
LSF_JOB_TYPE	Type of the job	String	LSF_JOB _TYPE="docker"
LSF_APP_NAME	Name of the application defined in lsb.applications	String	LSF_APP_NAME="boa"
LSF_JOB_ENV	Environment variable for LSF job	all, var1=value1, var2=value2	LSF_JOB_ENV="all"
LSF_JOB_DEFAULT _MEM_REQ	Percentage of physical memory required for each LSF job	Number	LSF_JOB_DEFAULT _MEM_REQ=50

Configuring NGINX

The BOA Appliance ships NGINX, which provides high performance reverse proxy and web server functionality. This section discusses the role of NGINX in BOA and the configuration choices you need to make when configuring NGINX for your environments

External Traffic Flows and NGINX:
All external traffic entering or exiting the BOA Appliance host goes through NGINX. External traffic here references communication flows between the BOA Appliance host and other components not residing on the BOA Appliance host.

The systems involved in these flows are:

- The BOA Appliance host where the BOA backend components (including NGINX) are running
- Web browsers on end user and administrator desktops and laptops
- The systems where the BOA SDK client-based experiment programs are running. These would typically be nodes on or with access to an HPC cluster

External traffic in the BOA system can be divided into several categories. These are highlighted in the following table:

Table 17.

External traffic	Description	Protocol	NGINX port
BOA server API traffic from BOA SDK client programs	These are the API calls made from the BOA SDK to the BOA server on behalf of client SDK programs. Examples include calls to create and run experiments on BOA server.	HTTP	nginx.http.port
		HTTPS	nginx.https.port
BOA server API traffic from BOA experiment viewer in browser	The experiment viewer is a JavaScript application running in a web browser. It invokes the BOA server API for operations like listing experiments for a user or triggering a sample experiment.	HTTP	nginx.http.port
		HTTPS	nginx.https.port
MQTT publish - subscribe topic notifications between BOA server and BOA SDK client programs	The BOA SDK subscribes to notifications from the BOA server for information such as parameters sets and explanation results which are published asynchronously as they are calculated by the BOA server.	MQTT over Web Sockets	nginx.http.port
		MQTT over Web Sockets with TLS	nginx.https.port
MQTT publish - subscribe topic notifications between BOA server and BOA experiment viewer in browser	These are the same sort of asynchronousnotifications as with the BOA SDK client programs. They allow the experiment viewer to display the latest experiment status and results as the optimizations are carried out on the BOA backend.	MQTT over Web Sockets	nginx.http.port
		MQTT over Web Sockets with TLS	nginx.https.port
BOA admin server API traffic from BOA admin UI in browser	The BOA admin user interface (UI) is a JavaScript application running in a web browser. Examples include the admin UI invoking the BOA admin server API to query operational status of the various services in BOA, to extract logs, and to trigger action against those services.	HTTP	nginx.http.port
		HTTPS	nginx.https.port

MQTT over Web Sockets puts an HTTP/HTTPS wrapper around the binary MQTT protocol. This makes it appropriate for configurations where network policies prohibit servers to expose any ports otherthan port 80 (HTTP) and 443 (HTTPS).

When planning your BOA deployment, you need to consider whether you want BOA external traffic to use HTTP, HTTPS, or both protocols.

In the NGINX portion of the BOA configuration file, there are two important configuration elements:

- The **nginx.http.port** element specifies the port number used for all HTTP and MQTT over web sockets external traffic.
- The **nginx.https.port** element specifies the port number used for all HTTPS and MQTT over web sockets with TLS external traffic.

If you want your external traffic to use HTTP, then you must open the port specified by **nginx.http.port** on the BOA appliance so that browser systems and systems running BOA SDK client can communicate over that port with the BOA appliance.

If you want your external traffic to use HTTPS, then you must open the port specified by **nginx.https.port** on the BOA appliance so that browser systems and systems running the BOA SDK client can communicate over that port with the BOA appliance.

The BOA Configuration Tool will configure NGINX so that the port identified by **nginx.https.port** uses TLS-based transport encryption.

BOA Appliance has no other port or protocol requirements other than for the standard HTTP and HTTPS ports.

Sample configuration

The following screen shows a sample of configuring NGINX for BOA:

```
nginx:
  http:
    port: 80
  https:
    port: 443
    ssl_policy: gen_self_signed
```

Element Description

The following table describes the job manager-related configuration elements.

Table 18. Description of the job manager-related configuration elements

Configuration element	Description	Format	Validation
nginx	Nests the set of elements associated with NGINX	N/A	N/A
nginx.http	Nests the set of elements associated with the NGINX HTTP port	N/A	N/A
nginx.http.port	Specifies the port exposed on NGINX for HTTP external traffic	Must be a valid TCP/IP port number	Validates that an integer value between 1 and 65535 is specified
nginx.https	Nests the set of elements associated with the NGINX HTTPS port	N/A	N/A

Table 18. Description of the job manager-related configuration elements (continued)			
Configuration element	**Description**	**Format**	**Validation**
nginx.https.port	Specifies the port exposed on NGINX for HTTPS external traffic	Must be a valid TCP/IP port number	Validates that an integer value between 1 and 65535 is specified
nginx.https.ssl_policy	Specifies whether BOA Configuration Tool should generate SSL/TLS certificates and key or you will supply your own. More details on SSL/TLS in the next section	gen_self_signed or cust_provided	Validates that one of the allowable policies is specified

Note: NGINX is always configured to expose both an HTTP port and an HTTPS port from the NGINX container to the BOA Appliance host. If you want to use HTTP only or HTTPS only, you can simply elect not to expose the other port using your OS firewall.

Configuring SSL/TLS

As discussed in the NGINX section, all external traffic in the BOA system is routed through NGINX over the ports specified with the **`nginx.http.port`** or **`nginx.https.port`** configuration elements.

The **`nginx.https.port`** is always configured for TLS-based transport encryption. This section discusses the options for configuring that encryption in BOA.

The **`nginx.https.ssl_policy`** configuration element specifies which option to use. The values of the **`nginx.https.ssl_policy`** you can specify are:

- **gen_self_signed** - Allow the BOA configuration tool to generate self-signed SSL/TLS certificates and keys which are then used to configure transport encryption on the port in NGINX.
- **cust_provided** – You supply your own SSL/TLS files which are then used to configure transport encryption on the port in NGINX.

Using SSL/TLS Files Generated by BOA:
When you specify a value of gen_self_signed for the nginx.https.ssl_policy configuration element, BOA Configuration Tool will generate the following files in your deployment under the BOA_INSTALL_DIR/deployment/<deployment-name>/config/ssl directory:

Table 19. File and usage	
File name	**Usage**
ca.key	This contains a private key generated for the CA. It is used to sign the CA certificate.
ca.crt	This contains the CA certificate file.
server.key	This contains a private key generated for the BOA server.
server.crt	This contains the server certificate for the BOA server. This certificate is signed by the generated ca.key.

Common Name and Subject Alternate Names:
The BOA server certificate is created with configuration information extracted from the boa_appliance.fqdn and boa_appliance.ip_address configuration elements. This section describes how these configurations contribute to the certificate.

Correct certificate verification requires that these values be accurate as they contribute to the common name and subject alternative names (SANs) used to create the certificate. By default these values are derived by invoking OS commands. You can override the values by modifying the boa_config.yml input file.

The X509 common name attribute is set from the value of the boa_appliance.fqdn configuration element.

A subject alternative name is added to the certificate for the IP address specified in the boa_appliance.ip_address configuration element.Using Your Own SSL/TLS Files If your organization already employs its own certificate authority for the generation of X509 certificate for use on your servers, you may already have existing certificate and key files with which you wihst ot configure transport encryption in BOA.

In order to do this, you need to:

- Specify a value of cust_provided for the nginx.https.ssl_policy configuration element.
- Copy your SSL/TLS files to your BOA deployment under the BOA_INSTALL_DIR/deployment/<deployment-name>/config/ssl directory using the same naming convention in the above table for your CA and server certificates and keys.

During configuration, the BOA Configuration Tool will validate the existence of the customer-provided files. However, no validation or verification of the customer-provided files is performed.

If your CA certificate includes one or more intermediate certificates, the certificate you provide under should be a CA bundle. A CA bundle is a file that contains root and intermediate certificates. The end-entity server certificate along with a CA bundle constitutes the certificate chain.

As with a CA certificate generated by the BOA Configuration Tool, you need to ensure that the CA certificate you provide for use in BOA is installed as a trusted CA certificate on each system running a BOA SDK client program or each system with a browser using the BOA Admin UI web application.

Configuring MongoDB

The BOA Appliance uses MongoDB to store its persistent data model. This includes information about experiments, results of the optimizations, explanation results for the experiments, and so on. This section details how MongoDB is configured.

Sample Configuration:

```
mongo:
  admin_user: mongoadmin
  admin_password: secret
  host_port: 27017
```

Element Descriptions

The following table describes the MongoDB-related configuration elements:

Table 20. MongoDB configuration element

Configuration elements	Description	Format	Validation
mongo	Nests the set of elements associated with MongoDB.	N/A	N/A
mongo. admin_user	The username of the administrative user for the BOA MongoDB. This is used to create an initial administrative user when the BOA MongoDB is initialized.	Valid MongoDB username.	None

Table 20. MongoDB configuration element (continued)

Configuration elements	Description	Format	Validation
mongo. admin_password	The password of the administrative user for the BOA MongoDB. This is used to create an initial administrative user when the BOA MongoDB is initialized.	Valid MongoDB password.	None
mongo. host_port	This is the port that is exposed on the BOA Appliance host. Changing this value does not impact how the internal BOA microservices connect to MongoDB. It does apply to how non-BOA external programs like backup utilities and MongoDB user interfaces connect to the BOA MongoDB instance.	Must be a valid TCP/IP port number.	Validates that an integer value between 1 and 65535 is specified.

Note:

- MongoDB database files are stored in a Docker volume.
- Unless you have a need to connect to the BOA MongoDB instance from systems outside the BOA Appliance host, the port specified in mongo.host_port need not be exposed to external systems via firewall rules.

Chapter 2. Administration Guide

These topics provide a list of the tasks you can perform to administer the IBM Bayesian Optimization Accelerator.

Log in to the administration GUI

You must log in with valid credentials to work in the BOA administration graphical user interface (GUI).

Logging in to the administration GUI

The user must use an existing username and password to log in to the administration GUI.

1. Open the administration GUI in your web-browser.
2. To log in to the administration GUI, enter the username and click the **Continue** button.
3. Enter the password and click the **Log in** button.

A new session is created which is valid for 24 hours. After login for the first time, the web-browser displays the configuration screen. Afterward, the web-browser displays the dashboard.

Reset admin login password

Admin user can reset or change the login password using a shell script as described below:

- Script name: reset_pwd_admin.sh
- Directory Path : /boabuild/installer/bin

Run the following command:

```
./reset_pwd_admin.sh
```

Note: This script will hit the API internally. Before running the script, update the port on to which BOA services are running. The default port value is 80.

Logging out of the administration GUI

You can log out from the administration GUI by clicking the **sign-out** option in the top navigation bar. After you click the log-out option, the current session will be expired.

Navigating the dashboard

The dashboard shows summary of User Statistics, Server Utilization, and BOA Application Services Status. You can also view summary of logs. You can click an icon or a label to go to the respective page. For example, if you click the **Users** label in the statistics section, the browser displays the Users page.

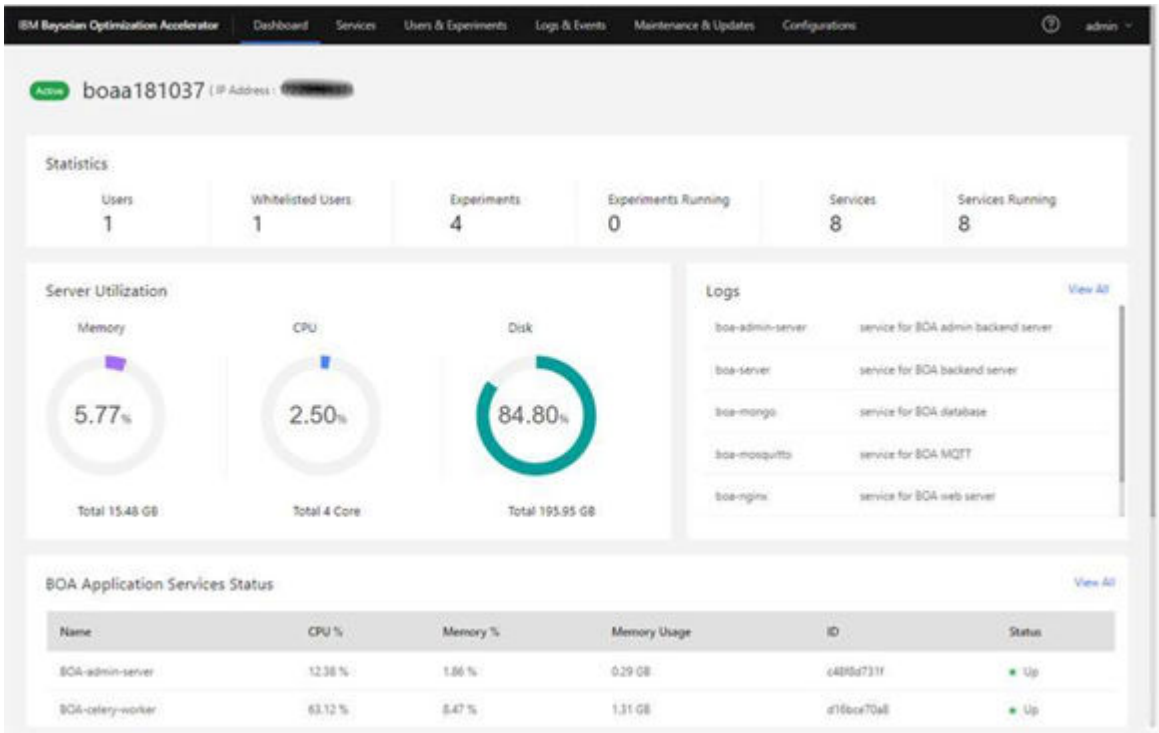

Figure 3. BOA administration dashboard

From the top navigation pane, you can go to the Services, User & Experiment, Logs and Events, Maintenance and Update, and Configuration pages.

Statistics

You can view summary statistics of Users, Whitelisted Users, Experiments, Experiments Running, Services, and Services Running.

Server Utilization

You can view status of Memory, CPU and Disk utilization of the server.

Logs

You can view summary of logs for each service. To view details of logs, click the **View All** label.

BOA Application Services Status

You can view details of CPU, Memory, Memory Usage, ID, and Status of all running services. To view details of services status, click the **View All** label.

Working in the administration GUI

From the top navigation pane, you can navigate to Services, User & Experiment, Logs and Events, Maintenance and Update, and Configuration pages.

Services

In the Services page, you can view the details of all services with their status (up or down), also other services related statistics. The details that are displayed in the page are: Name, CPU %, Memory %, Memory Usage, ID, Net I/O, Block I/O, Status.

- To restart all services:
  1. Click the **Restart All** button.
  2. A message box prompts you for the confirmation, click the **OK** button. All services restarts. The graphical user-interface (GUI) services that you are using also restarts, and the web-browser displays the Login page.
- To stop all services:
  1. Click the **Stop All** button.
  2. A message box prompts you for the confirmation, click the **OK** button. All the running services stops. The graphical user-interface (GUI) services that you are using also stops and the web-browser displays the Login page.

Note: If you stop all services, all the services along with this admin interface will be stopped, and you need to re-start all the services from the CLI. To restart all services, you can run the `boa_service_up.sh` shell script in the `/boabuild/installer/bin` directory. Run the script from the directory which contains the `docker-compose.yml` file. Run the following command:

```
./boa_service_up.sh
```

Users & Experiments

In the Users and Experiments page, you can view users and experiments details. You can view statistics of Total Users, Total Experiments, Running Experiments, and Completed Experiments. Also, you can view details of all registered users in the tabular format with their Name, User Id, Total Experiments, Completed, In Progress, Failed experiments.

You can also navigate to view and add White-listed Users page.

- Click the **View & Add Whitelisted User** label to navigate to the Whitelisted User page.
- To whitelist a user:
  1. Click the **Add user** button. The web-browser displays a modal window.
  2. Enter the email address of the user, then click the **Add** button. The email address is white-listed.

     Note: White-listed users include the registered users and existing white-listed users.

Logs & Events

In the Logs and Events page, you can view the list of services and their brief description. You can also view the raw logs individually by clicking on the **view log** link. Also, you can download all the service log files in the form of zipped folder by clicking the **download logs** button.

- To view details of a raw log or event, click the **View Logs** link. The web-browser displays the logs details page. This page displays the latest one thousand lines of logs of a service.
- To download all the services logs files in the form of zipped folder, click the **Download Logs** button.

Maintenance & Updates

In the Maintenance & Update page, you can set maintenance mode On or off. You can also set maintenance message which will be visible on the Experiment Viewer for the notification to all the BOA users. To switch On the maintenance mode, it is necessary that a message should be set already.

- To set the maintenance mode On or Off, admin has to toggle the switch. According to the mode, the message will be displayed above the toggle switch.
- To set the message, write a message in the Message box, then click the **Set Message** button. Once the message has been set, the **Set Message** will get converted to **Update Message**. Then the message will be displayed in the Experiment Viewer.
- To remove the message, the maintenance mode should be off, then only the user will be able to click on the **Remove Message** button otherwise the button will be disable in the case of maintenance mode is on.

Configuration

You can view the default BOA configurations for MongoDB, MQTT, LSF and Web Server Settings in the Configuration page. These configuration details are read only.

Chapter 3. BOA SDK Guide

This is a python client software development kit (SDK) for IBM Bayesian Optimization Accelerator.

The BOA SDK has been designed to be simple to use, but flexible in the range of configurations available for tailoring the optimization. This is achieved using the BOaaSClient object, which facilitates all communication with the BOA server. The optimization configuration is handled via a Python dictionary (or equivalently a JSON file).

Package	`boa_python`
Module	`sdk`
Submodule	`BOaaSClient`

SDK API

BOaasClient construction and initialization

Set the universal resource locator (URL) for the BOA instance. This is a prerequisite for the API. The URL is placed as the `BOaaSClient` object. The following screen displays the `BOaaSClient` object format:

```
boa_instance = BOaaSClient(host)
```

- The **`host`** element is a valid URL to the running BOA instance
- The **`boa_instance`** is returned object to represent the BOA instance

The user commands

This topic describes the method of registering a new user, log in a user, and log out a user by using the BOA API.

Register a new user with BOA

```
boa_instance.register(user_dict)
```

The register call maps to the REST API call `/users/register`. The register call takes in a dictionary of key-value pairs that includes the following information:

Table 21. User dictionary

Key	Value (description)
`_id`	Email address of the user
`name`	Full name of the user
`password`	Password for the user
`confirm_password`	Password again to confirm

Log in a user

```
user_token = boa_instance.login(user_dict)
```

This element authenticates a registered user. The login call maps to the REST API call `/users/login`. The login call takes in a dictionary of key-value pairs that includes the following information:

Table 22. User dictionary

Key	Value (description)
`_id`	Email address of the user
`password`	Password for the user

A token is returned from the login request, which must be retained to authenticate future requests.

Logout a user

```
boa_instance.logout(user_dict)
```

This element revokes a login token of a registered user. The logout call maps to the REST API call `/users/logout`. The logout call takes in a dictionary of key-value pairs that includes the following information:

Table 23. User dictionary

Key	Value (description)
`_id`	Email address of the user
`token`	An authentication token received from the login event

Experiment configuration and execution

This topic describes configuring query, create an experiment, Experiment dictionary, domain dictionary, Gaussian process kernel dictionary, sampling function dictionary, initialization type dictionary, random initialization type dictionary, stop an experiment, delete an experiment, get the next parameter set, response dictionary, sample status dictionary, add a new observation, and run the BOA appliance.

Configuration Query

```
boa_instance.config()
```

This element queries the running instance for its configuration. This returns the supported kernels and samplers of the BOA API. The config call maps to the REST API `call/config`.

Create an Experiment

```
experiment_id = boa_instance.create_experiment(user_dict, experiment_dict)
```

This element create a new Experiment for a registered user with the specified configuration. The `create_experiment` call maps to the REST API call `/experiments/create`. The `create_experiment` call takes in a dictionary of key-value pairs that includes the following information:

Table 24. User dictionary

Key	Value (description)
`_id`	Email address of the user
`token`	An authentication token received from login

Experiment dictionary

The Experiment dictionary is made up of a name, optimization type, and a set of dictionaries that define which kernels and sampling functions make up the Experiment.

Table 25. Experiment dictionary

Key	Value (description)
`name`	Experiment name
`optimization type`	`min` or `max`
`domain`	Experiment domain dictionary
`gaussian_process`	Gaussian process kernel dictionary
`sampling_function`	Sampling function dictionary
`initializaiton`	Initialization type dictionary

Domain Dictionary

Experiments must contain one domain per parameter.

Table 26. Domain dictionary

Key	Value (description)
name	Experiment name
min	Minimum value of the design space
max	Maximum value of the design space
step	Incremental value of the design space

Gaussian process kernel dictionary

This dictionary is used to configure the Gaussian process kernel that is used by the Experiment. Kernels have unique IDs and can be retrieved by using the `config()` command.

Table 27. Gaussian process kernel dictionary

Key	Value (description)
kernel_func	Specifies how the GP models the correlation between different points in the data set
scale_y	Specifies whether to scale the outputs when fitting the GP
scale_x	Specifies whether to scale the input (or x) values.

Probabilistic Backpropagation neural network (PBP) dictionary

Table 28. Gaussian process kernel dictionary

Key	Value (description)
batch_size	Sets the mini-batch size for the PBP network
scale_y	Specifies whether to scale the outputs when fitting the GP
scale_x	Specifies whether to scale the input (or x) values.

Sampling function dictionary

This dictionary is used to configure the sampling function of the Experiment. Sampling functions have unique IDs and can be retrieved by using the `config()` command.

Table 29. Sampling function dictionary

type	Specifies what type of acquisition function to use
epsilon	This variable controls the degree to which the optimizer will tend to exploit known 'good' areas of the domain (low epsilon), or favor exploring less well-known areas of the domain (high epsilon).
optimize_acq	Set to True.
outlier	Used only for the 'adaptive' acquisition functions
bounds	Contains the upper and lower bound for each parameter in the list.
scale	To scale the either output or input values

Initialization type dictionary

This dictionary is used to configure the initialization type of the Experiment. Kernels have unique IDs and can be retrieved by using the `config()` command.

Table 30. Initialization type dictionary

Key	Value (description)
`type`	`random` or `observations`
`random`	Random dictionary

Random initialization type dictionary

Table 31. Random initialization type dictionary

Key	Value (description)
`no_samples`	Number of the random samples
`seed`	Defines the seed element

Stop an Experiment

```
boa_instance.stop_experiment(experiment_id, user_token)
```

This element stops a specified experiment for a registered user. The `stop_experiment` call maps to the REST API call `/experiments/stop`.

Delete an Experiment

```
boa_instance.delete_experiment(experiment_id, user_token)
```

This element deletes a specified experiment for a registered user. The `stop_experiment` call sends a DELETE to the REST API call `/experiments/with_token`.

Get the next parameter set

```
response_dict = boa_instance.next_parameter_set(experiment_id, user_token)
```

This element retrieves and evaluates the next parameter set of an experiment. Before retrieve and evaluate an experiment, the experiment must be initialized. The `next_parameter_set` call maps to the REST API call `/experiments/next_parameter_set`. The `next_parameter_set` call is blocked until the observation is completed.

Response dictionary

Table 32. Response dictionary

Key	Value (description)
`next_parameter_set`	Parameter set dictionary
`sampler_status`	Sampler status dictionary

Sample status dictionary

Table 33. Sample status dictionary

Key	Value (description)
`initialized`	True or false on the sample initialized
`number_of_iterations`	The number of completed iterations
`generating_prior`	True or false

Add observation

```
boa_instance.new_observation(experiment_id, new_obsv_dict, user_token)
```

This element inserts a new observation for a specific experiment into the database. Before inserting a new observation, you must initialize the experiment. The `new_observation` call maps to the REST API call `/experiments/new_observation`. The `new_observation` call triggers a new iteration.

Run BOA

```
boa_instance.run(experiment_id, user_token, evaluation_function, constraint_function, explain,
no_epochs)
```

To run BOA, complete the following steps:

1. Get the next parameter set to try.
2. Evaluate y by using the specified evaluation function.
3. Insert the new observation into BOA until the wanted number of iterations is reached. The `evaluation_function` and `constraint_function` elements are user-defined.

Camelback example

The camelback example is a two-dimension optimization problem that optimizes the six-hump camelback function.

This function has global minima.

```
f(x) = -1.0316 at x = (0.0898, -0.7126) and (-0.0898, 0.7126)
```

Command-line input to run camelback_example.py

```
usage: python camelback_example.py [-h] [--hostname BOA_SERVER]
                           [--protocol {http, https}] [--port PORT]
                           [--epochs EPOCHS] [--ca-cert-path CA_CERT_PATH]

optional arguments:
  -h, --help            show this help message and exit
  --hostname BOA_SERVER, -ho BOA_SERVER
                        Hostname or IP address of BOA server to connect to.
                        (default: localhost)
  --protocol {http, https}, -pr {http, https}
                        Protocol to use to connect to BOA server. (default:
                        http)
  --port PORT, -p PORT  Port to connect to on BOA server. (default: 80)
  --epochs EPOCHS, -e EPOCHS
                        Number of epochs to perform. (default: 40)
  --ca-cert-path CA_CERT_PATH, -c CA_CERT_PATH
                        Path to the CA certificate file to use for https
                        protocol. (default: None)
```

Sample Camelback example python file

A sample camelback example is shown below:

```
"""
Licensed Materials - Property of IBM
“Restricted Materials of IBM”

5765-R17

© Copyright IBM Corp. 2020  All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with IBM Corp
"""

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import os, sys
from common.boa_examples_utils import BoaExamplesUtils
from boaas_sdk import BOaaSClient
import signal

example_description="""
        This example demonstrates basic BOA usage for a simple optimization problem.

        The BOA SDK has been designed to be simple to use, but flexible in the range of
        configurations available for tailoring the optimization. This is achieved using
        the BOaaSClient object, which facilitates all communication with the BOA server.
        The optimization configuration is handled via a Python dictionary (or
        equivalently a JSON file).
```

```
        The camelback example is a two dimensional optimization problem that optimizes
        the six humped camelback function. This function has global minima
        f(x) = -1.0316 at x = (0.0898, -0.7126) and (-0.0898, 0.7126))
        """

"""
Perform parsing of common commandline arguments shared by all BOA examples.
"""
args = BoaExamplesUtils.parse_commandline_args(example_description,default_epochs=40)
hostname = BoaExamplesUtils.get_connection_url(args)
print("Connecting to host: {}".format(hostname))
boaas = BOaaSClient(host=hostname,ca_cert_path=args.ca_cert_path)

def camelback6(x):
    # Six-hump camelback function
    x1 = x[0]
    x2 = x[1]
    f = (4 - 2.1 * (x1 * x1) + (x1 * x1 * x1 * x1) / 3.0) * (x1 * x1) + x1 * x2 + (-4 + 4 * (x2 *
x2)) * (x2 * x2)
    return f


experiment_config = {
    "name": "Reg camelback function",
    "domain": [
        {
            "name": "x1",
            "min": -2,
            "max": 2,
            "step": 0.01
        }, {
            "name": "x2",
            "min": -1,
            "max": 1,
            "step": 0.01
        }
    ],
    "model": {"gaussian_process": {
        "kernel_func": "Matern52",
        "scale_y": True,
        "scale_x": False,
        "noise_kernel": True,
        "use_scikit": True
    }},
    "optimization_type": "min",
    "initialization": {
        "type": "random",
        "random": {
            "no_samples": 3,
            "seed": None
        }
    },
    "sampling_function": {
        "type": "expected_improvement",
        "epsilon": 0.03,
        "optimize_acq": False,
        "outlier": False,
        "bounds": None
    }

}

user = {"_id": "boa_test@test.com", "password": "password"}
user_login = boaas.login(user)

if user_login == None:
    user = {"_id": "boa_test@test.com", "name": "BOA Test",
            "password": "password", "confirm_password": "password"}
    boaas.register(user)
    user_login = boaas.login(user)

print(user_login)
user_token = user_login["logged_in"]["token"]
print("user token")
print(user_token)
create_exp_user_object = {"_id": user["_id"], "token": user_token}
experiment_res = boaas.create_experiment(create_exp_user_object, experiment_config)
print(experiment_res)
experiment_id = experiment_res["experiment"]["_id"]
```

```
def signal_handler(signal, frame):
    """
    Function to handle Ctrl+C signal
    """

    print("Aborting...")
    boaas.abort(experiment_id, user_token)
    sys.exit()


# Catch ctrl+c signal to invoke its signal handler
signal.signal(signal.SIGINT, signal_handler)

boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6,
no_epochs=args.epochs, explain=False)
best_observation = boaas.best_observation(experiment_id, user_token)
print("best observation:")
print(best_observation)
boaas.stop_experiment(experiment_id=experiment_id, user_token=user_token)
```

Authenticate a registered user

The login call maps to the REST API call `/users/login user_login = boaas.login(user)`. A sample of the authenticate a registered user object is displayed in the following example.

```
if user_login == None:
    user = {"_id": "boa_test@test.com", "name": "BOA Test",
            "password": "password", "confirm_password": "password"}

    # Register a new user. The Register call maps to REST API call /users/register
    boaas.register(user)

    user_login = boaas.login(user)

print(user_login)
user_token = user_login["logged_in"]["token"]
print("user token")
print(user_token)
create_exp_user_object = {"_id": user["_id"], "token": user_token}
```

Authenticate a registered user

The login call maps to the REST API call /users/login user_login = boaas.login(user). A sample of the authenticate a registered user object is displayed in the following example.

```
if user_login == None:     user = {"_id": "boa_test@test.com", "name": "BOA Test",
            "password": "password", "confirm_password": "password"}

    # Register a new user. The Register call maps to REST API call /users/register
    boaas.register(user)

    user_login = boaas.login(user)
print(user_login)
user_token = user_login["logged_in"]["token"]
print("user token") print(user_token)
create_exp_user_object = {"_id": user["_id"], "token": user_token}
```

Create an experiment for a registered user with the specified configuration

The `create_experiment` call maps to REST API call `/experiments/create`. A sample of the create an experiment for a registered user with the specified configuration object is displayed in the following example.

```
experiment_res = boaas.create_experiment(create_exp_user_object, experiment_config)

print(experiment_res)
experiment_id = experiment_res["experiment"]["_id"]
```

Run BOA

Get the next parameter set to try, and evaluate y by using the specified evaluation function. Then, insert the new observation into BOA until the wanted number of iterations has been reached. A sample of the run BOA object is displayed in the following example.

```
boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6, no_epochs=4,
explain=False)
```

Print the best minimum or maximum observation of all the observations set

This API call maps to the REST API call `/experiments/best_pbservation`. A sample of the print the best minimum or maximum observation of all the observations set object is displayed in the following example.

```
best_observation = boaas.best_observation(experiment_id, user_token)
print("best observation:")
print(best_observation)
```

Stop an experiment for a registered user

The `stop_experiment` call maps to the REST API call `/experiments/stop`. A sample of the stop an experiment for a registered user object is displayed in the following example.

```
boaas.stop_experiment(experiment_id=experiment_id, user_token=user_token)
```

The following screen displays a sample output of running a BOA experiment:

{'_id': 'boa_test@test.com', 'logged_in': {'status': True, 'token': '5c806a36-2b01-11eb-ab4c-a9d04ca0275f'}, 'user': {'_id': 'boa_test@test.com', 'logged_in': {'status': True, 'token': '5c806a36-2b01-11eb-ab4c-a9d04ca0275f'}, 'name': 'boa_test', 'role': 'expuser', 'whitelist': 'true'}}
user token
5c806a36-2b01-11eb-ab4c-a9d04ca0275f
{'experiment': {'_id': '92c7a0be-2b01-11eb-a3b0-f9d9ecbd5771', 'created_at': 1605857132245, 'domain_id': '92c7a3b6-2b01-11eb-a3b0-f9d9ecbd5771', 'initialization': {'random': {'no_samples': 3, 'seed': None}, 'type': 'random'}, 'model': {'gaussian_process': {'kernel_func': 'Matern52', 'noise_kernel': True, 'scale_x': False, 'scale_y': True, 'use_scikit': True}}, 'n_sample': 1, 'name': 'Reg camelback function', 'next_parameter_set': {'X': [[]]}, 'observations_id': '92c7a348-2b01-11eb-a3b0-f9d9ecbd5771', 'optimization_type': 'min', 'sampler_config': {'generating_prior': False, 'mqtt': {'error_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/error', 'host': '172.20.181.37', 'lsf_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/lsf', 'new_explanation_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/new_explanation', 'new_observation_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/new_observation', 'new_y_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/new_y', 'next_constraint_X_function_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/next_constraint_X_function', 'next_constraint_X_function_notifications_topic_ui': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/next_constraint_X_function_ui', 'next_constraint_X_model_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/next_constraint_X_model', 'next_parameter_set_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/next_parameter_set', 'port': '80', 'sampler_status_notifications_topic': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771/sampler_status', 'subscription_topic_base': '92c7a1f4-2b01-11eb-a3b0-f9d9ecbd5771', 'url': 'mqtt://172:80', 'ws_https_url': 'wss://172:443/mqttws/', 'ws_port': '80', 'ws_url': 'ws://172:80/mqttws/'}, 'status': 'initializing'}, 'sampling_function': {'batch_sampling': {'use': False}, 'bounds': None, 'epsilon': 0.03, 'explain': None, 'optimize_acq': False, 'outlier': False, 'type': 'expected_improvement'}, 'type': 'default', 'user_id': 'boa_test@test.com', 'y_best': None}, 'status': 'Experiment created'}
Attempting to connect to message broker at URL ws://172:80/mqttws/...
Connected to message broker with result code 0
Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Calculating the next y value for suggested parameters...

Completed 40 iterations of Bayesian Optimisation.
40 iterations of Bayesian Optimisation have been completed in total for this experiment.
best observation:
{'best': {'X': [-0.12999999999999834, 0.7100000000000015], 'y': -1.0252309320636674}}

Chapter 4. User Guide

Getting Started with BOA

Common Optimization Conceptual Terminology

- **Design Variables (parameters)**: Design Variables are the parameters that are chosen to describe the design of a system. Objective function and constraints must be expressed as a function of design variables (or design vector X). Also, these variables are “controlled” by the designers.
- **Objective Function**: An objective function is the output that you want to maximize or minimize. It is what we will measure designs against to decide which option is best. The objective function can be thought of as the goal of your generative design process.
- **Constraints**: The constraints represent some functional relationships among the design variables and other design parameters satisfying certain physical phenomenon and certain resource limitations. The nature and number of constraints to be included in the formulation depend on the user. Constraints may have exact mathematical expressions or not.
- **Design space**: Placing the design variables along orthogonal axes defines a design space, or set of all possible design options. Each point in the design space corresponds to a chosen design.

Creating a user account

Whitelisted Users can register themselves on Experiment viewer UI and through SDK client. Admin user can only whitelist the users using the user whitelisting feature of the Admin GUI.

Running a sample example

For information on running sample experiment with BOA Python SDK, refer Annexure - II

Experiment configuration

BOA uses a configuration JSON file or an equivalent Python dictionary to configure the optimization. This topic describes the parameters used for configuration.

name
The name of your optimization experiment.

domain
The domain is the set of parameters that you want to search through to find the optimum parameters. For an engineering problem this might be the list of possible designs for a component, such as the engine of a car. For a chemical manufacturing problem, this could be the list of possible combinations of ingredients.

There are three ways to define the domain:

Grid
To define the domain as a grid, we specify the name of each parameter, a minimum value, a maximum value, and a step size. For example:

```
"domain": [
   {
     "name": "x1",
     "min": -2,
     "max": 2,
     "step": 0.01
   }, {
     "name": "x2",
     "min": -1,
```

```
      "max": 1,
      "step": 0.01
    }
    ]
```

Domain upload

Defines the domain as a simple list of parameter values that you want to search over. For example, you could use NumPy to generate a list of random two-dimensional parameters:

```
python domain = np.random.rand(500, 2).tolist() "domain": domain
```

Bounds

Defines the upper and lower bound for each parameter, without a step size. This is useful for problems where you do not want to generate all of the parameter values to try. To use bounds, define the domain using only the parameter names. For example:

```
python "domain": [ { "name": "x1" }, { "name": "x2" } ]
```

You must also set the `bounds` and `optimize_acq` fields on the `sampling_function` field.

bounds

Contains the upper and lower bound for each parameter in the list.

optimize_acq

Set to `True`.

Example:

```
python "sampling_function": { "optimize_acq": True, "bounds": [[-2,2],[-1,1]], ...
       }
```

model

Defines the surrogate model to use, which can be one of these options:

Gaussian Process (GP)

The standard for Bayesian optimization. The configuration for a GP is as follows:

```
"model":{"gaussian_process": {
  "kernel_func": "Matern52",
  "scale_y": True,
  "scale_x": False,
  "noise_kernel": True,
  "use_scikit": True,
  "optimizer": "LBFGS"
   }}
```

kernel_func

Specifies how the GP models the correlation between different points in the data set. BOA currently supports the following kernel functions: `SquaredExponential`, `Matern32`, and `Matern52`.

scale_y

Specifies whether to scale the outputs when fitting the GP, and whether to use scaled values when computing the acquisition function. If you choose not to scale the output and we have very large magnitude values, BOA may prematurely terminate optimization. It is generally recommended that this parameter is set to `True`.

scale_x

Specifies whether to scale the input (or x) values. It is generally recommended that this is set to `False`.

noise_kernel

Specifies whether to include a noise kernel in our GP. When set to `True`, a white noise kernel is added to the GP in order to model noise in our data. This is best practice for most real-world optimization problems, as there is almost always noise in real-world data.

use_scikit

Allows switching between the IBM GP Pro GP library and the open source scikit-learn GP. This is useful for debugging issues between different GP libraries.

use_scikit=True indicates BOA uses scikit-learn Gaussian Process Algorithm

use_scikit=False indicates BOA uses IBM Gaussian Process PRO library

Note: PTF1 for Fix Pack 1.1.0.1 only supports use_scikit as True, which supports only open source scikit-learn Gaussian Process Algorithm.

The following are the important considerations have been taken to the PTF1 for Fix Pack 1.1.0.1 are given below:

- If use_scikit=True, then BOA experiment will run as it is expected.
- If use_scikit = False, then BOA experiment will stop and throws an exception shown below:
  - This version of BOA does not support use_scikit as False. Please set use_scikit as True in experiment config and re-run the experiment by setting the use_scikit as True, you will be using Gaussian Process Model of Scikit Learn library.

If use_scikit parameter is not provided in the BOA experiment configuration, then it considers as default value of use_scikit which is True.

optimizer

Allows you to select the optimizer used for fitting the GP hyper-parameters. Supported values are: LBFGS (default), `Adam`, SGD, `Adadelta`, `Adagrad`, ASGD, and `RMSprop`.

Probabilistic Backpropagation neural network (PBP)

A probabilistic neural network that is useful for problems that involve high dimensionality or a lot of data. Use the following to configure PBP:

```
"model":{"PBP": {
  "scale_y": True,
  "scale_x": False,
  "batch_size": 32
   }}
```

batch_size

Sets the mini-batch size for the s network. A larger mini-batch size can speed up training for the network, but may negatively impact its generalisation. This parameter is used solely for PBP and has nothing to do with batch sampling and parallel optimization.

The batch_size should be less than or equal to no_of_random (or initial observations, if initialization type is Observation) i.e. batch_size<=no_of_random samples otherwise PBP model won't work.

scale_y

Specifies whether to scale the outputs when fitting the GP, and whether to use scaled values when computing the acquisition function. If you choose not to scale the output and we have very large magnitude values, BOA may prematurely terminate optimization. It is generally recommended that this parameter is set to `True`.

scale_x

Specifies whether to scale the input (or x) values. It is generally recommended that this is set to `False`.

initialization

Specifies how to initialize the optimizer. There are two options: initialization by random samples (`random`) or by uploading observations (`observations`). Random initialization randomly selects values from the domain, whereas the observation-based initialization allows you to specify a list of parameter values to initialize the optimizer.

Example of specifying random initialization:

```
"initialization": {
    "type": "random",
```

```
    "random": {
      "no_samples": 3,
      "seed": None
    }
  },
```

no_samples
Specifies the number of samples to use for initialization.

seed
Specifies whether to set the NumPy random seed for the initialization. Setting the seed means you can start with the same initial samples each time. This can be useful for reproducibility or debugging issues with the optimization, but generally this should be set to None.

Example of using observations for initialization:

```
"initialization": {
    "type": "observations",
    "observations": obs
  },
```

observations
In this example, the observations (obs) is a list where the first element is a list of parameters (inputs), and the second element is a list of responses (outputs). For example:

```
obs = [[[0.342234, 0.543677], [1.1464377, 0.535677], [1.324556, 0.934786]],
       [0.435561, 0.5342, 0.532566]]
```

sampling_function
Specifies how the optimizer will sample from the domain.

type
Specifies what type of acquisition function to use. The acquisition function is core to how Bayesian optimization functions, and different acquisition functions will result in different optimizer behaviors.

For standard optimization problems, the Expected Improvement (EI) acquisition function is typically recommended. Another commonly used acquisition function is Probability of Improvement (PI). Both of these acquisition functions have an `epsilon` variable associated with them.

It is typically advised to use one of the following acquisition functions: `expected_improvement`, `adaptive_expected_improvement`, `probablity_improvement`, or `adaptive_probability_improvement`.

BOA optimizer also supports, epsilon_greedy, maximum_entropy and random_sampler sampler types but only when design variables domain is defined as Grid (Discrete variables).

epsilon
Some acquisition functions have an associated epsilon variable. This variable controls the degree to which the optimizer will tend to exploit known 'good' areas of the domain (low epsilon), or favor exploring less well-known areas of the domain (high epsilon).

BOA provides a further enhancement on this in the form of its adaptive acquisition functions, which dynamically tune the epsilon value to automatically trade-off between exploration and exploitation.

scale
Depending on the version of BOA being used, the `scale` parameter may not be required. If required, this should always be set to `False`.

outlier
Used only for the 'adaptive' acquisition functions. If this is set to 'True', outliers are removed before calculating the new epsilon value.

constraints
Used for optimization with constraints. For details, see “Optimization with constraints” on page 55.

optimize_acq
Set to `True`.

bounds
Contains the upper and lower bound for each parameter in the list.

When using batch (or parallel) optimization, a different set of acquisition functions should be used. See 'Batch Optimization' and the 'hyper_parameter_tuning_batch.py' example.

Example `sampling_function` configuration:

```
"sampling_function": {
  "type": "adaptive_expected_improvement",
  "epsilon": 0.03,
  "optimize_acq": False,
  "outlier": False,
  "bounds": None,
  "scale": False
}
```

explain
Defines the explainability features computed for BOA. If the explain field isn't used, BOA will run without explainability. PBP model doesn’t support Explanation i.e. explain should be False in the experiment configuration for all the experiments running with model: PBP:

feature_importance
Whether to compute the feature importance. Values are `True` and `False`

feature_interaction
A list of feature interactions to use, one or both of PDP and `H_statistic` can populate the list.

features_idx
A list.

Example `explain` field:

```
"explain": {
    "feature_importance": True,
    "feature_interaction": ['PDP', 'H_statistic'],
    "features_idx": [0,1]
}
```

batch_sampling

use
Whether to use batch sampling. Values are `True` or `False`.

batch_size
The batch size or maximum batch size, depending on type of batch acquisition function used)

type
The type of batch acquisition function used. Values are:

BatchThompsonSampler
batch Thompson sampler

KMeansSampler_EI
k-means clustered expected improvement batch sampler

qEI_CLSampler
Constant Liar batch sampler

qEI_BasicSampler
Basic Batch Sampler

B3Sampler_EI
Budgeted Batch Sampler

qEI_KBSampler
Kriging Believer Sampler

Example:

```
"batch_sampling": {
          "use": True,
          "batch_size": 5,
          "type": "BatchThompsonSampler"
      },
```

See `hyper_parameter_tuning_batch.py` for a complete example of how to use batch sampling with BOA.

Recommend Batch Sampler with BOA optimizer is BatchThompson. All Batch Samplers do not support domain defined as Bounds (continuous design variables).

safe_initialization
Specifies what happens if the GP optimization fails. This can occur for a number of reasons, such as a poor choice of kernel or a poor set of initial observations.

True
An error is returned if the GP hyper parameter optimization fails and BOA stops running. This is the default.

False
A warning is raised if the GP hyper parameter optimization fails, BOA collects a new random sample of initial observations and tries to run again. The number of retries is set by the `max_tries` parameter.

The reason for not retrying by default is that BOA will often be used with expensive evaluations. Thus, if this were the default behavior, users could incur significant additional cost if BOA kept trying to regenerate its initial observations upon failure. Typically, the GP fitting can be addressed by changing the choice of kernel. Thus, careful consideration is advised before setting `safe_initialization` to `True`.

max_tries
When `safe_initialization` is `True`, specifies how many times BOA collects a new random sample of initial observations and re-run GP fitting.

optimization_type
Specifies whether optimization is minimization (`min`) or maximization (`max`).

n_sample
Set to 1.

optimizer
Selects the optimizer used if using `bounds`. This is different from the `optimizer` parameter associated with the `gaussian_process` field. When using `bounds`, the acquisition function computed via optimization, rather than by evaluating values over a domain grid. Therefore, it requires a second global optimizer within BOA. The choices of optimizer are: `direct`, `cobyla`, and `basinhopping`.

This parameter is only used if `bounds` has a value other than `None`, and `optimize_acq` is `True`.

Note: Cobyla optimizer is not supported in PTF1 for Fix Pack 1.1.0.1.

Multiple objective optimization
These fields are used only for multiple objective optimization. For more information, see “Multiple objective optimization” on page 53.

task_weights
How much weight to attribute to each task.

combi_ops
Which operation to use when combining the task information during optimization. This can be either `sum` or `prod` for each task.

samplers

This is populated with a list, each item of which is a dictionary (or JSON object) defining a sampler object for each independent task, or objective. As such, each sampler dictionary (or JSON object) contains the following configuration fields: `name`, `optimization_typ`, `model`, `initialization`, and `sampling_function`, as described above.

Example:

```
experiment_config = {
  "name": "Multi Object",
  "type": "multi_object",
  "domain": [
  {
    "name": "x1",
    "min": -2,
    "max": 2,
    "step": 0.01
  }, {
    "name": "x2",
    "min": -1,
    "max": 1,
    "step": 0.01
  }
  ],
  "task_weights": [1,1],
  "combi_ops": ["prod", "prod"],
  "n_sample": 1,
  "samplers": [
  {
    "name": "Camelback",
    "optimization_type": "min",
    "model":{"gaussian_process": {
      "kernel_func": "Matern52",
      "scale_y": True,
      "scale_x": True,
      "noise_kernel": False
    }},
    "sampling_function": {
      "type": "expected_improvement",
      "epsilon": 0,
      "optimize_acq": False,
      "outlier": False,
      "bounds": None,
      "scale": False,
      "print_level": False,
      "run_epsilon": 0
    },
    "initialization": {
      "type": "random",
      "random": {
        "no_samples": 3,
        "seed": None
      }
    }
  },
  {
    "name": "Shifted Camelback",
    "optimization_type": "min",
    "model": {"gaussian_process": {
      "kernel_func": "Matern52",
      "scale_y": True,
      "scale_x": True,
      "noise_kernel": False
    }},
    "sampling_function": {
      "type": "probability_improvement",
      "epsilon": 0.03,
      "optimize_acq": False,
      "outlier": False,
      "bounds": None,
      "scale": False,
      "print_level": False,
      "run_epsilon": 0
    },
    "initialization": {
      "type": "random",
      "random": {
        "no_samples": 3,
        "seed": None
```

```
      }
    }
  }
  ]
}
```

Multi-objective optimization doesn't support domain defined as Bounds (continuous variables). Hence, bounds: None should be used, and Domain should be defined as Grid.

See `camelback_example_multi-objective.py` for a working example of using BOA for multiple objective optimization.

Complete configuration example

The following is an example BOA configuration for use with the BOA Python SDK:

```
experiment_config = {
    "name": "Reg camelback function",
      "domain": [
      {
        "name": "x1",
        "min": -2,
        "max": 2,
        "step": 0.01
      }, {
        "name": "x2",
        "min": -1,
        "max": 1,
        "step": 0.01
      }
      ],
    "model":{"gaussian_process": {
    "kernel_func": "Matern52",
    "scale_y": True,
    "scale_x": False,
    "noise_kernel": True,
    "use_scikit": True
     }},
    "optimization_type": "min",
    "initialization": {
      "type": "random",
      "random": {
        "no_samples": 3,
        "seed": None
      }
    },
    "sampling_function": {
    "type": "expected_improvement",
    "epsilon": 0.03,
    "optimize_acq": False,
    "outlier": False,
    "bounds": None,
    "scale": False
  },
    "n_sample": 1

}
```

Batch sampling

For many real-world optimization problems, you can evaluate several candidate parameters in parallel. Thus, some optimization algorithms can work with batches of parameters - for example, by suggesting a batch of 10 new sets of parameters to try, rather than just a single set of parameters.

To work with these algorithms, BOA supports batch sampling. When using batch sampling, BOA receives multiple outputs, or responses, from the function it is optimizing and uses information from all of these responses to formulate the next batch of parameters to try.

To use batch sampling, use the `batch_sampling` field in the BOA configuration:

batch_sampling

use
Whether to use batch sampling. Values are `True` or `False`.

batch_size
The batch size or maximum batch size, depending on type of batch acquisition function used)

type
The type of batch acquisition function used. Values are:

BatchThompsonSampler
batch Thompson sampler

KMeansSampler_EI
k-means clustered expected improvement batch sampler

qEI_CLSampler
Constant Liar batch sampler

qEI_BasicSampler
Basic Batch Sampler

B3Sampler_EI
Budgeted Batch Sampler

qEI_KBSampler
Kriging Believer Sampler

Example:

```
"batch_sampling": {
          "use": True,
          "batch_size": 5,
          "type": "BatchThompsonSampler"
      },
```

See `hyper_parameter_tuning_batch.py` for a complete example of how to use batch sampling with BOA.

Recommend Batch Sampler with BOA optimizer is BatchThompson. All Batch Samplers do not support domain defined as Bounds (continuous design variables).

See hyper_parameter_tuning_batch.py for a complete example of how to use batch sampling with BOA.

Multiple objective optimization

You might want to optimize more than a single objective. For example, you might be developing a new drug for which you want to optimize its potency, but also want to ensure that it can be manufactured economically. Thus, you also want to optimize the cost. This allows you to find the trade-off between the best compound and the associated cost of manufacture.

The BOA multiple objective optimization functionality allows you to do this. Multiple object optimization shares the same configuration foundation as standard optimization, but duplicates some of the fields so that you can define the optimization configuration for each objective individually.

Multi-objective optimization doesn't support domain defined as Bounds (continuous variables).Hence, "bounds": None should be used, and Domain should be defined as Grid.

These fields are used only when setting up multiple objective optimization:

task_weights
How much weight to attribute to each task.

combi_ops
Which operation to use when combining the task information during optimization. This can be either `sum` or `prod` for each task.

samplers

This is populated with a list, each item of which is a dictionary (or JSON object) defining a sampler object for each independent task, or objective. As such, each sampler dictionary (or JSON object) contains the following configuration fields: `name`, `optimization_typ`, `model`, `initialization`, and `sampling_function`, which are described in “Experiment configuration” on page 45.

Example:

```
experiment_config = {
  "name": "Multi Object",
  "type": "multi_object",
  "domain": [
  {
    "name": "x1",
    "min": -2,
    "max": 2,
    "step": 0.01
  }, {
    "name": "x2",
    "min": -1,
    "max": 1,
    "step": 0.01
  }
  ],
  "task_weights": [1,1],
  "combi_ops": ["prod", "prod"],
  "n_sample": 1,
  "samplers": [
  {
    "name": "Camelback",
    "optimization_type": "min",
    "model":{"gaussian_process": {
      "kernel_func": "Matern52",
      "scale_y": True,
      "scale_x": True,
      "noise_kernel": False
    }},
    "sampling_function": {
      "type": "expected_improvement",
      "epsilon": 0,
      "optimize_acq": False,
      "outlier": False,
      "bounds": None,
      "scale": False,
      "print_level": False,
      "run_epsilon": 0
    },
    "initialization": {
      "type": "random",
      "random": {
        "no_samples": 3,
        "seed": None
      }
    }
  },
  {
    "name": "Shifted Camelback",
    "optimization_type": "min",
    "model": {"gaussian_process": {
      "kernel_func": "Matern52",
      "scale_y": True,
      "scale_x": True,
      "noise_kernel": False
    }},
    "sampling_function": {
      "type": "probability_improvement",
      "epsilon": 0.03,
      "optimize_acq": False,
      "outlier": False,
      "bounds": None,
      "scale": False,
      "print_level": False,
      "run_epsilon": 0
    },
    "initialization": {
      "type": "random",
      "random": {
        "no_samples": 3,
        "seed": None
```

```
        }
      }
    }
    ]
}
```

Optimization with constraints

A common scenario is optimizing some function given a set of constraints on the function. For example, say you are developing a new kind of detergent and you never want the cost of ingredients to exceed a given amount. You can use constraint-based optimization to find the best combination of parameters that fulfills your cost requirements.

BOA lets you perform optimization with constraints with either *hard* or *soft* constraints on the objective function. Using hard constraints means that the parameter set is rejected if the constraint criteria are not met. Using soft constraints gives the optimization a little more flexibility to find a balance between the optimal set of parameters and the constraint criteria.

When using constraints, you must do the following:

- In the configuration, set the `type` field to `constraints`.
- Provide the constraint function to the BOaaSClient.run() call.

The constraint function should contain code that evaluates the parameters and returns either a 1 or 0. If the constraint criteria apply and the set of parameters is not suitable, the function should return a 1. Otherwise, the function should return a 0, as the parameters are valid and thus not 'constrained'.

Here is an example constraint function:

```
def constraint_func(x):
    c = 0
    if x[0][0] > 0:
        c = 1
    return c
```

In the above, if element `x[0][0]` is greater than 0, the function returns 1, indicating that this set of parameters should not be considered. This is added to the BOaaSClient.run() call; passing it to the `constraint_f`:

```
boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6,
          constraint_f=constraint_func, no_epochs=10, explain=False)
```

See `camelback_constraints_example.py` for a working example of optimization with constraints in BOA.

Note: PTF1 for Fix Pack 1.1.0.1 does not support a use case where in all the initial samples for an experiment provided by the of BOA optimizer violates the constraint condition. In this case, BOA throws an error.

Troubleshooting

Below is the troubleshooting guide for the common issues which we can observe while using BOA.

Stop_experiment API

It is advisable to execute stop_experiment() API after every experiment completion. This will help in releasing the resources being used by the completed experiment. BOA experiments performed through SDK should use stop_experiment() API. Experiments executed from Experiment Viewer have to be stopped explicitly. Please refer BOA SDK guide for the usage of stop_experiment API.

Connection Refused error

If you are getting this error while running the experiment through SDK, please verify the host ip address of BOA server provided in the command while executing the experiment with –hostname xxx.xx.xx.xx.

Also, please verify the port which you have provided i.e. –port. BOA server should run the host and port which you have provided.

JSON Decode error

If you are running the experiment on CLI and getting the below error:

```
simplejson.errors.JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

Above error indicates that there is some issue with the command line arguments which you should have passed while running the experiment. For more information, please refer Camelback Example under BOA SDK GUIDE.

BOA Experiment Client Keep Alive implementation:

IF user is executing long running experiment through BOA SDK and somehow the connection of experiment client gets disconnected with the BOA server due to BOA server goes down, then user has to start their experiment from beginning again. To tackle such user scenarios, we are sharing a keep alive example code snippet that can be used for long running experiment.

Replace the below line-

```
boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6,
no_epochs=args.epochs, explain=False)
```

With this function implementation

```
def check_server_connection(): """
```

Function to check for BOA server connection when BOA server is down, to keep client alive:

```
"""
    connection_attempt = 0
    def wait_for(min_s=1, max_s=360000, increase=1):
        """
        Iterator which yields i value until it is lesser than max
        """
        i = min_s
        while i <= max_s:
            yield i
            i += increase
    # This will make boaas.run to run in loop if in case of server disconnection
    for seconds in wait_for():
        # If BOA server is up and connected, it will go to try block
        # If boaas.run throws error due to connection lost with server
        # It will go to except block and try to reconnect to server
        try:
            boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6,
no_epochs=args.epochs, explain=False)
        except Exception as exception:
            connection_attempt += 1
            print(repr(exception)+"Error occurred. Trying to reconnecting to server... ")
            time.sleep(seconds)
    # If server is still not reachable after above waiting period then stop this experiment
with the below error message
    msg = "Reconnection failed after {} connection attempts".format(connection_attempt)
    raise ConnectionError(msg)
# Calling this function to reconnect to server in case of outage in between the experiment run
check_server_connection()
```

Delete BOA experiments from CLI

To delete an experiment from CLI, you can use the delete experiment shell script which will be helpful in cleaning up the BOA system by releasing the resources used by these unwanted experiments. The following is the details of the script:

- Script name: delete_exp_script.sh
- Directory Path : /boabuild/installer/bin

To delete a BOA experiment, run the following command:

```
/delete_exp_script.sh
```

This script will expect user to provide the below inputs:

- Id of experiment which user wants to delete
- User token which was used during the experiment
- BOA server IP address and Port

Disconnection Between LSF Host and BOA Server

Whenever the connection between LSF host and BOA server (LSF client) gets disconnected, do the following workaround steps to bring back BOA server to the normal running conditions.

You can identify the issues, with the BOA experiments are failed and below error statements are appearing in LSF LIM daemon log file stored at /home/share/lsf/log/lim.log.master

Intercluster request from host <172.xxx.x.x:xxxx> not using privileged port

Received request <5> from non-LSF host 172.xxx.x.x:xxxxx

1. Delete BOA experiments which are not started from MongoDB:

   Do the following steps to delete the BOA experiments which are not in the "completed" state and with “0” iterations.

   - SSH to BOA host server machine with valid user credentials
   - Run the following command on a terminal to get boa-mongo service container ID.

     ```
     docker ps | grep boa-mongo
     ```

   - An output similar to the following screen is displayed on the successful execution of the command:

     ```
     dbd01b3bf4eb              boa/mongo:1.1.0.1-ppc64le           "docker-entrypoint.s…"
     27017/tcp                                       1-1-0-1_boa-mongo_1
     ```

   - Run the following command to open bash terminal for "boa-mongo" container.

     ```
     docker exec -it <boa-mongo_container_id> /bin/bash
     ```

   - Run the following command to open mongo shell inside the “boa-mongo” container.

     ```
     mongo
     ```

     An output similar to the following screen is displayed on the successful execution of the command:

     ```
     MongoDB shell version v3.6.3
             connecting to: mongodb://127.0.0.1:27017
             MongoDB server version: 3.6.3
             Welcome to the MongoDB shell.
             For interactive help, type "help".
             For more comprehensive documentation, see
                     http://docs.mongodb.org/
             Questions? Try the support group
                     http://groups.google.com/group/mongodb-user
             MongoDB Enterprise
     ```

   - Run the following set of commands on the mongo shell to delete the experiments which are not started. You can collect the MongoDB user credentials from the administrator.

     ```
     use boa
         db.auth('<mongodb_username>','<mongodb_password>')
         db.experiments.deleteMany({"$and": [
                 {"sampler_config.status": {"$not": {"$in": ["completed"]}}},
                 {"sampler_config.number_of_iterations": 0}]}
     ```

An output similar to the following screen is displayed on the successful execution of the command:

```
switched to db boa
1
acknowledged" : true, "deletedCount" : 23
```

2. LSF Reconfiguration:

   To reconfigure the LSF, do the following steps:

   - Run the following command to reconfigure LSF daemons, then accept the prompts.

     ```
     lsadmin reconfig
     ```

   - An output similar to the following screen is displayed on the successful execution of the command:

     ```
         Checking configuration files ...
                     No fatal errors found.
                      Warning: Some configuration parameters may be incorrect.
                     They are either ignored or replaced by default values.
                       There are warning errors.
             Do you want to see the detailed messages? [y/n] n
             No fatal errors found.
             Warning: Some configuration parameters may be incorrect.
                      They are either ignored or replaced by default values.
             Do you want to reconfigure? [y/n] y
             Restart only the master candidate hosts? [y/n] y
             Restart LIM on <boaasrv181018> ...... done
     ```

   - Run the following command to restart mbatchd.

     ```
     badmin mbdrestart
     ```

     An output similar to the following screen is displayed on the successful execution of the command:

     ```
     Checking configuration files ...
             No errors found.
             mbatchd parallel restart initiated
     ```

3. Cleaning up LSF jobs:

   It is recommended to kill all the jobs in the LSF cluster. To kill the jobs in the LSF Cluster, do the steps that follows:

   - Run the following command to switch to LSF administrator user.

     ```
     su lsfadmin
     ```

   - Run the following command to kill all the jobs

     ```
     bkill -r 0
         Wait for bkill execution to get completed
     ```

4. Go to BOA installation directory, $BOA_INSTALL_DIR/deployment/default and run the following command

   ```
   docker-compose down
   ```

5. Manual clean-up of /etc/hosts:

   Whenever BOA services are restarted during maintenance windows or failure scenarios, a duplicate entry will get created in the /etc/hosts file. These duplicate entries may introduce disconnection between the BOA services and LSF. BOA administrator should manually remove these duplicate entries using the below mentioned steps:

   - Login as BOA administrator to the BOA appliance.

- Review the “boa_config.yml” placed at path $BOA_ROOT_DIR/installer/samples/.
- Stop the BOA micro-services from the Admin GUI.
- Take note of the setting specified for “docker_subnet”
- For example, docker_subnet: "172.178.0.0/16"
- Remove entries from the “/etc/hosts” file for any IP Addresses that match the subnet mask defined in the “docker_subnet” configuration property.
- For example, if your “docker_subnet” property is "172.178.0.0/16", you would remove /etc/hosts entries starting with 172.178.0.x.
- After updating the “/etc/hosts” file, you can restart BOA services.

  Note: The /etc/hosts file must only be updated only when the BOA microservices are in “Stopped” state.

6. Go to BOA installation directory, $BOA_INSTALL_DIR/deployment/default and run the following command

```
docker-compose up -d
```

Annexure I

Working sampler type list

Table 34. Working sampler type list

Sampler	Sampler type	Domain type	Initialization type
Normal Sampler	expected_improvement	Grid	Random
			Observation
		Bounds	Random
			Observation
	probability_improvement	Grid	Random
			Observation
		Bounds	Random
			Observation
	adaptive_expected_improvement	Grid	Random
			Observation
		Bounds	Random
			Observation
	adaptive_probability_improvement	Grid	Random
			Observation
		Bounds	Random
			Observation
	epsilon_greedy	Grid	Random
			Observation
		Bounds	Random
			Observation
	maximum_entropy	Grid	Random
			Observation
		Bounds	Random
			Observation
	Random	Grid	Random
			Observation
	multi_sampler	Grid	Random
			Observation
	Constraints	Grid	Random
			Observation

Table 34. Working sampler type list (continued)

Sampler	Sampler type	Domain type	Initialization type
Batch Sampler	Batch Thompson	Grid	Random
			Observation
	KMeansSampler	Grid	Random
			Observation
	qEI_BasicSampler	Grid	Random
			Observation
	qEI_CLSampler	Grid	Random
			Observation
	B3Sampler_EI	Grid	Random
			Observation
	qEI_KBSampler	Grid	Random
			Observation

Tested and Supported Spectrum LSF Version

BOA MVP deliverable has been tested with Spectrum LSF 10.1 FP 9 (10.1.0.9).

Annexure II

Camelback example batchThompson

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import os, sys
from common.boa_examples_utils import BoaExamplesUtils
from boaas_sdk import BOaaSClient

example_description="""
        This example demonstrates basic BOA usage for a simple optimization problem.

        The BOA SDK has been designed to be simple to use, but flexible in the range of
        configurations available for tailoring the optimization. This is achieved using
        the BOaaSClient object, which facilitates all communication with the BOA server.
        The optimization configuration is handled via a Python dictionary (or
        equivalently a JSON file).

        The camelback example is a two dimensional optimization problem that optimizes
        the six humped camelback function. This function has global minima
        f(x) = -1.0316 at x = (0.0898, -0.7126) and (-0.0898, 0.7126))
        """

"""
Perform parsing of common commandline arguments shared by all BOA examples.
"""

args = BoaExamplesUtils.parse_commandline_args(example_description,default_epochs=40)
hostname = BoaExamplesUtils.get_connection_url(args)
print("Connecting to host: {}".format(hostname))
boaas = BOaaSClient(host=hostname,ca_cert_path=args.ca_cert_path)

def camelback6(x):
    # Six-hump camelback function
    x1 = x[0]
    x2 = x[1]
    f = (4 - 2.1 * (x1 * x1) + (x1 * x1 * x1 * x1) / 3.0) * (
            x1 * x1) + x1 * x2 + (-4 + 4 * (x2 * x2)) * (x2 * x2)
    return f


experiment_config = {
    "name": "Reg camelback function",
    "domain": [
        {
            "name": "x1",
            "min": -2,
            "max": 2,
            "step": 0.01
        }, {
            "name": "x2",
            "min": -1,
            "max": 1,
            "step": 0.01
        }
    ],
    "model": {"gaussian_process": {
        "kernel_func": "Matern52",
        "scale_y": True,
        "scale_x": False,
        "noise_kernel": True,
        "use_scikit": False
    }},
    "optimization_type": "min",
    "initialization": {
        "type": "random",
        "random": {
            "no_samples": 3,
            "seed": None
        }
    },
    "sampling_function": {
        "type": "expected_improvement",
        "epsilon": 0.03,
        "optimize_acq": False,
        "outlier": False,
        "bounds": None,
        "batch_sampling": {
```

```
            "use": True,
            "batch_size": 1,
            "type": "BatchThompsonSampler"
        }
    }

}

user = {"_id": "boa_test@test.com", "password": "password"}
user_login = boaas.login(user)

if user_login == None:
    user = {"_id": "boa_test@test.com", "name": "BOA Test",
            "password": "password", "confirm_password": "password"}
    boaas.register(user)
    user_login = boaas.login(user)

print(user_login)
user_token = user_login["logged_in"]["token"]
print("user token")
print(user_token)
create_exp_user_object = {"_id": user["_id"], "token": user_token}
experiment_res = boaas.create_experiment(create_exp_user_object, experiment_config)
print(experiment_res)
experiment_id = experiment_res["experiment"]["_id"]


def signal_handler(signal, frame):
    """
    Function to handle Ctrl+C signal
    """

    print("Aborting...")
    boaas.abort(experiment_id, user_token)
    sys.exit()


# Catch ctrl+c signal to invoke its signal handler
signal.signal(signal.SIGINT, signal_handler)


boaas.run(experiment_id=experiment_id, user_token=user_token, func=camelback6,
no_epochs=args.epochs, explain=False)
best_observation = boaas.best_observation(experiment_id, user_token)
print("best observation:")
print(best_observation)
boaas.stop_experiment(experiment_id=experiment_id, user_token=user_token)
```

Chapter 5. BOA Fix Tool

Program Temporary Fix 1 (PTF1) for Fix Pack 1.1.0.1 includes a new tool "BOA Fix Tool" for applying, rolling back, and listing BOA PTF Packages.

PTF1 for Fix Pack 1.1.0.1 Deployment Using BOA Fix Tool

This topic describe the steps to deploy the PTF1 for Fix Pack 1.1.0.1 on your BOA appliance server.

1. Create a temporary directory for unpacking the PTF1 for Fix Pack 1.1.0.1 and set an environment variable for the directory.

   ```
   export BOA_PTF1_TEMPDIR=/tmp/
   PTF1 mkdir ${BOA_PTF1_TEMPDIR}
   ```

   Note: Here we used the temporary directory as "/tmp/PTF1".
2. Download the PTF1 for Fix Pack 1.1.0.1 to the BOA_PTF1_TEMPDIR. The package is named as " boa-1.1.0.1-PTF-01-external-050120211101.tgz”.
3. Unpack the PTF1 for Fix Pack 1.1.0.1.

   ```
   cd ${BOA_PTF1_TEMPDIR}
   tar -xzvf boa-1.1.0.1-PTF-01-external-050120211101.tgz
   tgz
   ```
4. • Create and export an environment variable for the root directory on your BOA Version 1.1.0.1 deployment. The PTF1 for Fix Pack 1.1.0.1 is deployed only on the deployment of BOA Version 1.1.0.1. The directory specified for this variable is the root of your BOA deployment. Under this directory there are subdirectories namely "deployments", "installer", "license" and "sdk".

   ```
   export BOA_ROOT_DIR=/opt/IBM/boa/1.1.0.1
   ```

   Note: Here we used root directory as BOA Version 1.1.0.1 installation root directory.
5. PTF1 for Fix Pack 1.1.0.1 includes a new tool namely BOA Fix Tool for applying, rolling back and the listing BOA PTF Fix Packages. You need to copy this tool to your BOA installation directory with the following command.

   ```
   cp ${BOA_PTF1_TEMPDIR}/boa_fix_tool.sh ${BOA_ROOT_DIR}/installer/bin
   ```
6. Before applying the PTF1 for Fix Pack 1.1.0.1, you need to stop the BOA infrastructure containers currently running on the BOA Appliance. To stop the services, run the following command:

   ```
   cd ${BOA_ROOT_DIR}/deployments/default
   docker-compose down
   ```
7. • Run the following command to apply the PTF1 for Fix Pack 1.1.0.1 on your BOA deployment.

   ```
   cd ${BOA_ROOT_DIR}/installer/bin
   ./boa_fix_tool.sh -a apply -p ${BOA_PTF1_TEMPDIR}/boa-1.1.0.1-PTF-01-SERVER-package.tgz
   ```
8. Run the following command to verify that the PTF1 for Fix Pack 1.1.0.1 is successfully applied on your BOA deployment. The output indicates that the Fix ID 01 is installed.

   ```
    cd ${BOA_ROOT_DIR}/installer/bin
   ./boa_fix_tool.sh -a list
   ```
9. PTF1 Fix pack 1.1.0.1 includes the patches for 3 LSF binaries ("bsub", "mbatchd" and "mbschd"). The LSF patches can be applied to the LSF Deployment using the below steps

   • Shutdown the LSF cluster by running the following command.

```
lsfshutdown
```

- Setup an environment variable that points to the root directory for your LSF installation on the BOA appliance and copy the LSF patched binaries

```
export LSF_ROOT=/home/share/lsf/10.1/linux3.10-glibc2.17-ppc64le/
cp ${BOA_PTF1_TEMPDIR}/lsf_patch/bsub ${LSF_ROOT}/bin/
cp ${BOA_PTF1_TEMPDIR}/lsf_patch/mbschd ${LSF_ROOT}/etc/
cp ${BOA_PTF1_TEMPDIR}/lsf_patch/mbatchd ${LSF_ROOT}/etc/
```

- To start the LSF cluster, run the following command.

```
lsfstartup
```

- You can validate the patch by validating the version of "bsub" on the master host.

```
bsub -V
```

  An output similar to the following screen is displayed on the successful execution of the command:

```
IBM Spectrum LSF 10.1 build 1234567
Copyright International Business Machines Corp. 1992, 2016.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA
ADP Schedule Contract with IBM Corp.
binary type: linux3.10-glibc2.17-ppc64le
notes: unoptimized for debugging purposes
fixes: Hot fix for BOA float client issue
```

10. To start the BOA infrastructure containers, run the following command.

```
cd ${BOA_ROOT_DIR}/deployments/default
docker-compose up -d
```

11. Your BOA Version 1.1.0.1 server deployment is successfully applied with the PTF1 for Fix Pack 1.1.0.1.

Rolling Back PTF1 for Fix Pack 1.1.0.1

This topic describe the steps to roll back the PTF1 for Fix Pack 1.1.0.1 from your BOA Appliance server.

1. Before you roll back the PTF1 for Fix Pack 1.1.0.1, you need to stop the BOA infrastructure containers currently running on the BOA Appliance. To stop the services, run the following command:

```
cd ${BOA_ROOT_DIR}/deployments/default
docker-compose down
```

2. To roll back the PTF1 for Fix Pack 1.1.0.1 from your deployment, run the following command.

```
cd ${BOA_ROOT_DIR}/installer/bin
./boa_fix_tool.sh -a rollback -s latest
```

3. Run the following command to verify that the PTF1 for Fix Pack 1.1.0.1 is successfully rolled back from your BOA deployment. The output indicates that no fix packages are installed.

```
cd ${BOA_ROOT_DIR}/installer/bin
./boa_fix_tool.sh -a list
```

4. To start the BOA infrastructure containers, run the following command.

```
cd ${BOA_ROOT_DIR}/deployments/default
docker-compose up -d
```

5. Your BOA Version 1.1.0.1 server deployment is successfully rolled back with the PTF1 for Fix Pack 1.1.0.1.

SDK Considerations of PTF1 for Fix Pack 1.1.0.1

PTF1 for Fix Pack 1.1.0.1 provides a complete refresh of the SDK component in file ${BOA_PTF1_TEMPDIR}/boa-1.1.0.1-PTF-01-SDK-package.tgz.

Note: BOA SDK Python Library is same as BOA Fix Pack 1.1.0.1. You do not need to refresh your BOA client environment for PTF1 for Fix Pack 1.1.0.1.

Though the BOA SDK Python Library is same as BOA Version 1.1.0.1, three BOA SDK example programs have been changed to adhere to the new policy for handling of use_scikit. The below three programs are included in file boa-1.1.0.1-PTF-01-SDK-package.tgz.

```
sdk/examples/camelback_example.py
sdk/examples/camelback_example_keep_alive.py
sdk/examples/hyper_parameter_tuning_example.py
```

Chapter 6. Release notes

These release notes provide an overview of what is new or changed in IBM Bayesian Optimization Accelerator (BOA) versions 1.1.0, 1.1.0.1, and PTF1 for Fix Pack 1.1.0.1

Contents

Enhancement and Issues Fixed in PTF1 for Fix Pack 1.1.0.1

IBM Bayesian Optimization Accelerator Fix Pack 1.1.0.1 has a number of issues. In addition to the BOA fixes, PTF1 Package also includes a BOA Fix Tool that is used to update the BOA Docker Images on the BOA appliance and patches 3 LSF binaries.

New features and enhancements in Fix Pack 1

Fix Pack 1

BOA 1.1.0.1 includes the following new features and enhancements.

- **Bytecode Obfuscation for plain text python files in BOA**

  The MVP release was delivered with the plain text python code. With Bytecode Obfuscation enhancement, plain text python files are being converted to python bytecode(.pyc) as part of BOA installer.

New features and enhancements in 1.1.0

BOA 1.1.0 includes the following new features and enhancements. For more information, follow the link, where provided.

- **Productize BOA AC922 Appliance (non-clustered) BOA micro-services orchestration via docker-compose**

  IBM Bayesian Optimization Accelerator is provided as-an-appliance in the MVP release. Along with the host pre-requisite software, BOA will be pre-installed and deployed on the AC922 Power hardware. BOA architecture comprises of multiple interacting micro-services which are deployed via docker-compose orchestrator in a single node configuration. All the message communication happening on the external BOA interfaces supports TLS encryption.

- **BOA Python SDK**

  This is the Python client software development kit (SDK) for IBM Bayesian Optimization Accelerator. It has been designed to be simple to use, but flexible in the range of configurations available for tailoring the optimization. This is achieved using the BOaaSClient object, which facilitates all communication with the BOA server. The optimization configuration is handled via a Python dictionary (or equivalently a JSON file).

- **LSF integration on BOA Appliance for BO jobs**

  Spectrum LSF 10.1 is used as a workload manager in the BOA appliance. It provides a comprehensive set of intelligent, policy-driven scheduling features that enables full utilization of the available compute infrastructure resources and ensures optimal application performance by running Bayesian Optimization jobs in a docker container on demand.

- **Interface functions**

BOA supports a concept of Interface functions. These are the primary interfaces meant for BOA interaction with external HPC simulation environments. As part of solving real-world problems, optimized sample data points evaluated by BOA are supplied via interface functions for running experiments on external simulators and receiving back the corresponding outputs for the simulation runs. Sample interface functions showcasing integration with OpenFOAM simulator are hosted on the repository links: https://github.com/IBM/boa-interface-functions/ and https://github.com/IBM/boa-interface-functions/tree/main/Code_Aster respectively.

- **BOA Admin GUI – Phase 1**

  Admin User Interface has been developed for Admin User where he can log in and can see the detailed information like Dashboard, Services, Logs, User Management, Maintenance Mode, co.

  – Login – Where Admin User logs in by entering the valid credentials and the first-time user lands on to Configuration page and second-time user will navigate to Dashboard.
  – Configuration - Where Admin User can see various configuration settings like Mongo settings, MQTT settings, LSF settings, and Web Server settings.
  – Dashboard – Shows the consolidated data for Admin GUI.
  – User & Experiments – Displays registered users and whitelisted users and their experiment running.
  – Services – Displays list of services with their status and an option to Restart and Stop All Services.
  – Maintenance Mode – The User can start, stop all the experiments and services with setting the custom message on Experiment GUI.
  – Logs – Displays list of services with their individual logs and provide option to download the consolidate data for all the services logs.

  For more details on the Admin User Interface, refer Knowledge Center link https://www.ibm.com/support/knowledgecenter/SS9TMV and https://www.ibm.com/support/knowledgecenter/SS9TMV_1.1.0/base/boa_admin.html.

- **Known defects**

  Below are the Open defects:

  – Issue with Constraint sampler throwing Type Error. PTF1 for Fix Pack 1.1.0.1 does not support a use case where in all the initial samples for an experiment provided by the of BOA optimizer violates the constraint condition and it throws errors.

    https://github.com/ibm-boa/IBM_dev/issues/43
  – Issue with cobyla optimizer. In experiment with domain as “bounds” Cobyla optimizer is not working when "use_scikit"= True.

    https://github.com/ibm-boa/IBM_dev/issues/95

- **Defect Fixed**

  Below defects have been fixed in the PTF1 for Fix Pack 1.1.0.1

  – Issue of Batch Sampling with type qEI_KBSampler and B3Sampler_EI fails with runtime error for optimizer jobs.

    https://github.com/ibm-boa/IBM_dev/issues/23
  – Issue of Error occurred during camelback_example_bounds with Epsilon Greedy sampler. BOA 1.1.0.1 (fix pack) does not support the Epsilon Greedy sampler for continuous optimization type (Domain type bound).

    https://github.com/ibm-boa/IBM_dev/issues/31
  – Issue of repeated sampling points. Few data points are repeated for direct bounds optimizer (continuous optimization) and few mean values are also observed in those data points.

    https://github.com/ibm-boa/IBM_dev/issues/36
  – Issue with basinhopping optimizer. Using basinhopping optimizer, it happens that the optimizer does not accept any value to calculate the parameter.

https://github.com/ibm-boa/IBM_dev/issues/50

- Issue with data Normalization while performing prediction. BOA optimization models are using normalized data for prediction in the experiment config. This results in wrong predicted values at times.

  https://github.com/ibm-boa/IBM_dev/issues/52

- Issue with camelback example with initial observations and domain list.py fails. BOA does not support File Upload Domain type with the LSF scheduler while performing the memory calculation.

  https://github.com/ibm-boa/IBM_dev/issues/61

Notices

This information was developed for products and services offered in the US.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_.

Privacy policy considerations

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as the customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details the section entitled "Cookies, Web Beacons and Other Technologies" and the "IBM Software Products and Software-as-a-Service Privacy Statement" at http://www.ibm.com/software/info/product-privacy.