Portal Gun Documentation

Release 0.2.0

Vadim Fedorov

Nov 30, 2018

Contents:

1 User Guide **3**
- 1.1 Overview . . . 3
- 1.2 Installation . . . 4
- 1.3 Configuration . . . 5
- 1.4 Portal Specification . . . 6
- 1.5 Commands . . . 12

Deep Learning on Amazon EC2 Spot Instances without the agonizing pain.

Release v0.2.0.

Portal Gun is a command line tool that automates repetitive tasks associated with the management of Spot Instances on Amazon EC2 service.

Primarily it is intended to simplify usage of AWS Spot Instances for Deep Learning. This focus will further shape the future development.

CHAPTER 1

User Guide

1.1 Overview

Portal Gun originates from the necessity to rent some GPU resources for **Deep Learning** and the natural aspiration to save some money. Of course, it might be useful in other use cases involving AWS Spot Instances.

Notice, though, that Portal Gun is not a generic tool. If you need full control over AWS resources from command line, use the AWS CLI instead.

1.1.1 Concepts

Portal

Portal Gun was design around the concept of *portals*, hence the name. A *portal* represents a remote environment and encapsulates such things as virtual server (Spot Instance) of some type, persistent storage, operating system of choice, libraries and frameworks, etc.

To *open* a portal means to request a Spot Instance. To *close* a portal means to cancel the request and terminate the instance. For example, if you are training a model, you open a portal for a training session and close it, when the training is finished. If you follow the recommended workflow (*see bellow*), you should be able to open the portal again and find everything exactly like you left it before.

A portal is defined by a *portal specification* file which describes a particular environment in JSON format.

Portal specification includes::

- characteristics of a Spot Instance to be requested (instance type, key pair for secure connection, security group, availability zone, etc.);
- software configuration (AMI, extra dependencies to be installed, etc.);
- persistent data storage (see bellow);
- data synchronization channels (see bellow).

Persistent Volume

AWS Spot Instance are volatile by nature, therefore, some external storage is needed to persist the data. The most efficient option is EBS Volume.

Portal Gun allows you to manage EBS Volumes from command line. It also automatically attaches and mounts volumes to instances according to the portal specifications. You might have a single volume to store everything (dataset, code, checkpoints of training, etc.) or use separate volumes for each type of data.

Channel

Channels can be defined in portal specification to synchronize files between a Spot Instance and your local machine. Synchronization is done continuously using `rsync` and should be started explicitly with a command. Every channel is either *inbound* (files are moved from remote to local) or *outbound* (files are moved from local to remote).

For instance, you may edit scripts locally and configure a channel to send them to the remote instance after every save. You might configure another channel to automatically get some intermediate results from the remote instance to your local machine for preview.

1.1.2 Typical Workflow

A typical Deep Learning workflow with Portal Gun is as follows:

1. Using Portal Gun create a new volume (e.g. named 'data') for all your data;
2. Configure a portal backed by the 'data' volume and a non-GPU instance;
3. Open the portal configured in step 2;
4. Connect to the non-GPU instance and copy all necessary data to the 'data' volume;
5. Close the portal configured in step 2;
6. Configure a portal backed by the 'data' volume and a GPU instance;
7. Open the portal configured in step 6;
8. Run training on the GPU instance;
9. Close the portal configured in step 6.

1.2 Installation

Portal Gun has the following external dependencies:

- boto3 - to make requests to AWS;
- Fabric - to execute commands over ssh;
- marshmallow - for serialization.

Note that Python 3 is not supported yet, because Fabric is Python 2 only. Migration to Python 3 should be made after the first stable release of Fabric 2.

1.2.1 Install or upgrade from the PyPI

It is **strongly recommended** to install Portal Gun in **a virtual Python environment**. For details about virtual environments see virtualenv documentation.

To install the latest stable version from the PyPI:

```
$ pip install -U portal-gun
```

To install the latest pre-release version from the PyPI:

```
$ pip install -U portal-gun --pre
```

1.3 Configuration

1.3.1 Application Config

Portal Gun reads basic configuration from a file in JSON format. By default it looks for a file named `config.json` in the following locations (in that order):

1. script running path
2. `~/.portal-gun/`

When Portal Gun is installed in a virtual Python environment (recommended), script running path is `/virtual-env-path/bin/`.

A custom location and filename may be specified using `-c, --config` argument.

Values to set in the configuration file:

```
{
        "aws_region": "current AWS region",
        "aws_access_key": "access key for your AWS account",
        "aws_secret_key": "secret key for your AWS account"
}
```

Credentials (access and secret keys) for programmatic access on behalf of your AWS account can be found in the IAM Console. **It is recommended to create a separate user** for programmatic access via Portal Gun.

1.3.2 AWS Access Rights

Portal Gun requires the following access rights:

```
iam:PassRole

ec2:DescribeAccountAttributes

ec2:DescribeAvailabilityZones
ec2:DescribeSubnets

ec2:CreateVolume
ec2:ModifyVolume
```

(continues on next page)

(continued from previous page)

```
ec2:AttachVolume
ec2:DetachVolume
ec2:DeleteVolume
ec2:DescribeVolumes
ec2:DescribeVolumeStatus
ec2:DescribeVolumeAttribute
ec2:DescribeVolumesModifications

ec2:RequestSpotFleet
ec2:CancelSpotFleetRequests
ec2:RequestSpotInstances
ec2:CancelSpotInstanceRequests
ec2:ModifySpotFleetRequest
ec2:ModifyInstanceAttribute
ec2:DescribeSpotFleetRequests
ec2:DescribeSpotInstanceRequests
ec2:DescribeSpotFleetInstances
ec2:DescribeSpotPriceHistory
ec2:DescribeSpotFleetRequestHistory
ec2:DescribeInstances
ec2:DescribeInstanceStatus
ec2:DescribeInstanceAttribute

ec2:CreateTags
ec2:DeleteTags
ec2:DescribeTags
```

IAM Policy is the most convenient way to grant required permissions. Create a new policy and attach it to a user which will be used for programmatic access via Portal Gun.

Reference policy granting required permissions can be found here. You can make it more strict, for instance, by limiting access right to a particular region.

1.3.3 Additional Resources

- Controlling Access Using Policies

1.4 Portal Specification

```
{
        "spot_instance": {
                "instance_type": "string (required)",
                "image_id": "string (required)",
                "key_pair_name": "string (required)",
                "identity_file": "string (required)",
                "security_group_id": "string (required)",
                "availability_zone": "string (required)",
                "subnet_id": "string (optional)",
                "ebs_optimized": "boolean (optional)",
                "remote_group": "string (required)",
                "remote_user": "string (required)",
                "python_virtual_env": "string (optional)",
                "extra_python_packages": [
```

(continues on next page)

(continued from previous page)

```
                        "string (optional)"
                ]
        },
        "spot_fleet": {
                "iam_fleet_role": "string (required)"
        },
        "persistent_volumes": [
                {
                        "volume_id": "string (required)",
                        "device": "string (required)",
                        "mount_point": "string (required)"
                }
        ],
        "channels": [
                {
                        "direction": "string (required)",
                        "local_path": "string (required)",
                        "remote_path": "string (required)",
                        "recursive": "boolean (optional)",
                        "delay": "float (optional)"
                }
        ]
}
```

Draft portal specification as the one above can be created using *init* command.

1.4.1 Schema

spot_instance

Type: object. Required.

Specification of a Spot Instance.

spot_instance . instance_type

Type: string. Required.

Type of AWS EC2 Instance to be requested.

Detailed description of available types can be found here.

spot_instance . image_id

Type: string. Required.

Id of Amazon Machine Images (AMI) to be used to launch the Instance.

For information about Deep Learning AMIa see *bellow*.

spot_instance . key_pair_name

Type: string. Required.

Name of AWS EC2 Key Pair to be used to connect to the Instance. The Key Pair should match the identity file.

spot_instance . identity_file

Type: string. Required.

Location of identity file (private key) to be used for authentication when connecting to the Instance. This file is generated by AWS upon creation of a new Key Pair. The identity file should match the Key Pair.

spot_instance . security_group_id

Type: string. Required.

Id of AWS EC2 Security Group to be associated with the Instance. Security Group controls which inbound and outbound traffic is allowed for the Instance. Make sure to at least allow inbound ssh traffic (port 22).

spot_instance . availability_zone

Type: string. Required.

Name of AWS Availability Zone in which to launch the Instance. Availability Zone has to be within the Region, specified in the *application config*.

Note that Spot Instance prices might differ between Availability Zones.

spot_instance . subnet_id

Type: string. Optional.

Id of Subnet to be used for the Instance. If not specified, default Subnet of the Availability Zone is used.

spot_instance . ebs_optimized

Type: boolean. Optional.

Enable/disable EBS Optimization. An EBS–optimized instance uses an optimized configuration stack and provides additional, dedicated capacity for EBS I/O.

spot_instance . remote_group

Type: string. Required.

Default AMI user group. For images based on Ubuntu in most cases the group will be *ubuntu*. If in doubt, check AMI usage instructions.

spot_instance . remote_user

Type: string. Required.

Default AMI username. For images based on Ubuntu in most cases the username will be *ubuntu*. If in doubt, check AMI usage instructions.

spot_instance . python_virtual_env

Type: string. Optional.

Default Python virtual environment to be used to install extra Python packages. Should be specified when *spot_instance.extra_python_packages* is specified.

spot_instance . extra_python_packages

Type: array of strings. Optional.

Extra Python packages to be installed in the default Python virtual environment.

spot_fleet

Type: object. Required.

Specification of a Spot Instance Fleet.

spot_fleet . iam_fleet_role

Type: string. Required.

IAM role that grants the Spot Fleet permission to terminate Spot Instances on your behalf when you cancel its Spot Fleet request. For instance:

arn:aws:iam::123456789012:role/aws-ec2-spot-fleet-tagging-role

where "123456789012" should be replaced by your AWS Account Id which can be found in AWS Console.

persistent_volumes

Type: array of objects. Required.

Specifications of EBS volumes to be attached. Use *volume* group of commands to manage and list volumes.

Note: to be able to attach EBS Volumes to an Instance, they should be in the same Availability Zone.

persistent_volumes[] . volume_id

Type: string. Required.

Id of EBS volume to be attached to the Instance.

persistent_volumes[] . device

Type: string. Required.

Name of device to represent the attached volume. For example, `/dev/xvdf`. See documentation for details.

persistent_volumes[] . mount_point

Type: string. Required.

Mounting point within the Instance file system, where device representing the volume should be mounted. For example, `/home/ubuntu/workspace` (assuming that AMI username is *ubuntu*).

channels

Type: array of objects. Required.

Specifications of file synchronization channels.

channels[] . direction

Type: string. Required.

Direction of file transfer. Expected values are "*in*" and "*out*". Inbound channel transfers files from the remote Instance to the local machine. Outbound channel transfers files from the local machine to the remote Instance.

channels[] . local_path

Type: string. Required.

Local path to be used in synchronization. Note that synchronization is done via `rsync`, therefore, similar rules regarding the trailing slash (/) in the source path are applied (see *excerpt* of rsync help for details).

channels[] . remote_path

Type: string. Required.

Remote path to be used in synchronization. Note that synchronization is done via `rsync`, therefore, similar rules regarding the trailing slash (/) in the source path are applied (see *excerpt* of rsync help for details).

channels[] . recursive

Type: boolean. Optional.

Enable/disable recursive synchronization. Disabled by default.

channels[] . delay

Type: float. Optional.

Delay between two consecutive synchronization attempts. Defaults to 1 second.

1.4.2 Additional Details

Deep Learning AMIs

Amazon Machine Images (AMI) are used to create virtual machines within the AWS EC2. They capture the exact state of software environment: operating system, libraries, applications, etc. One can think of them as templates. Pre-configured AMIs can be found in AWS Marketplace. Some of them are free to use, others have per hour license price depending on the set of pre-installed software. EC2 users can also created their own Images.

Deep Learning AMIs - is a group of AMIs created by Amazon specifically for deep learning applications. They come pre-installed with open-source deep learning frameworks including TensorFlow, Apache MXNet, PyTorch, Chainer, Microsoft Cognitive Toolkit, Caffe, Caffe2, Theano, and Keras, optimized for high performance on Amazon EC2 instances. These AMIs are free to use, you only pay for the AWS resources needed to store and run your applications. Official documentation, guides and tutorials can be found here.

There are several different flavors of Deep Learning AMIs. Check the guide to know the difference between them.

In order to instruct Portal Gun to use one of the Deep Learning AMIs to create an AWS Instance you need to know its **ID**:

1. Go to AWS Marketplace and search for *"deep learning ami"*;
2. Pick an image from the search results, e.g. *Deep Learning Base AMI (Ubuntu)*;
3. On the AMI's page click **Continue to Subscribe** button;
4. On the opened page select **Manual Launch** tab;
5. In the **Launch** section you will see AMI IDs for different regions.

Rsync Help on Trailing Slash

An excerpt of `man rsync`:

Recursively transfer all files from the directory src/bar on the machine foo into the /data/tmp/bar directory on the local machine:

```
$ rsync foo:src/bar /data/tmp
```

A trailing slash on the source changes this behavior to avoid creating an additional directory level at the destination:

```
$ rsync foo:src/bar/ /data/tmp
```

You can think of a trailing / on a source as meaning "copy the contents of this directory" as opposed to "copy the directory by name", but in both cases the attributes of the containing directory are transferred to the containing directory on the destination. In other words, each of the following commands copies the files in the same way, including their setting of the attributes of /dest/foo:

```
$ rsync /src/foo /dest
$ rsync /src/foo/ /dest/foo
```

1.5 Commands

Print top-level help message:

```
$ portal -h
```

Add `-h` (or `--help`) flag after commands and command groups to print corresponding help messages. For instance, print help message for the `volume` group including the list of commands:

```
$ portal volume -h
```

Top-level command options:

`-c` `CONFIG`, `--config` `CONFIG`
Set name and location of configuration file.

1.5.1 Persistent Volumes

This section documents a group of commands that are used to manage persistent volumes. For information on how to configure attachment of persistent volumes to instances see *Portal Specification* section.

Create

Create a new EBS volume:

```
$ portal volume create
```

Every volume requires **size** (in Gb) and **availability zone** to be specified. **Name** is optional, but recommended. If these three properties are not set using the command options, they will be requested from the standard input.

Upon successful creation of a new volume its `<Volume-Id>` will be provided.

Command options:

`-n` `NAME`, `--name` `NAME`
Set name for new volume.

`-s` `SIZE`, `--size` `SIZE`
Set size (in Gb) for new volume.

`-z` `ZONE`, `--zone` `ZONE`
Set availability zone for new volume.

`-S` `SNAPSHOT`, `--snapshot` `SNAPSHOT`
Set Id of a snapshot to create new volume from.

`-t` `key:value [key:value ...]`, `--tags` `key:value [key:value ...]`
Set user tags for new volume.

List

List existing EBS volume:

```
$ portal volume list
```

By default `list` command outputs only the volumes created by Portal Gun on behalf of the current AWS user. To list all volumes use `-a` flag.

Command options:

`-a, --all`
Show all volumes, not only ones created by Portal Gun.

Update

Update an AWS volume:

```
$ portal volume update <Volume-Id>
```

Command options:

`-n` NAME, `--name` NAME
Update name of volume.

`-s` SIZE, `--size` SIZE
Update size of volume.

`-t` key:value [key:value ...], `--tags` key:value [key:value ...]
Add user tags for volume.

Delete

Delete an AWS volume:

```
$ portal volume delete <Volume-Id>
```

By default `delete` command deletes only the volumes created by Portal Gun on behalf of the current AWS user. To force deletion of a volume use `-f` flag.

Command options:

`-f, --force`
Delete any volume, even not owned.

1.5.2 Portals

Portal is the main concept of the Portal Gun (see *Concepts* for details).

Init

Create a draft *portal specification* file:

```
$ portal init <Portal-Name>
```

A file with the name `<Portal-Name>.json` will be created. Modify this file to set the appropriate values (see *Portal Specification* section).

Open

To open a portal means to request and configure a Spot Instance according to the *portal specification*. Open a portal:

```
$ portal open <Portal-Name>
```

Ssh

Once the portal is opened, connect to the remote instance via ssh:

```
$ portal ssh <Portal-Name>
```

For long-running tasks like training a model it is particularly useful to be able to close current ssh session without interrupting the running task. One way of achieving this is offered by `tmux`. "It lets you switch easily between several programs in one terminal, detach them (they keep running in the background) and reattach them to a different terminal." - tmux wiki. You can run `tmux` within ssh session and then run the long task within `tmux` session. Portal Gun allows you to use tmux session automatically with `-t` command option.

Command options:

`-t` `[session]`, **`--tmux`** `[session]`
: Automatically open tmux session upon connection. Default session name is *portal*.

Info

Check information about a portal:

```
$ portal info <Portal-Name>
```

Information includes portal status (open or closed). If portal is open, information about the instance and attached volumes is provided.

When Portal Gun is used in a shell script, it might be useful to get specific bits of information without the rest of the output. In this case use command option `-f` to get the value of one particular field. Supported fields are:

- name - portal name;
- status - portal status (open or close);
- id - instance id;
- type - instance type;
- user - remote user;
- host - remote host;
- ip - public IP of instance;
- remote - user@host
- key - local ssh key file

For instance, to copy a file from remote instance to local machine you can use Portal Gun to look up connection details:

```
$ scp -i "`portal info <Portal-Nane> -f key`" `portal info <Portal-Nane> -f remote`:/
↪path/to/file /local/folder/
```

Command options:

-f FIELD, **--field** FIELD
Print value for a specified field (name, status, id, type, user, host, ip, remote, key).

Close

To close a portal means to cancel a Spot Instance request and terminate the instance itself. Close a portal:

```
$ portal close <Portal-Name>
```

1.5.3 Channels

Channels are used to sync remote and local folders. A channel has direction, source and target folders, and other properties. Every channel belongs to a portal and should be configured in the corresponding portal specification file (see *Portal Specification* section for details).

Channel

Start syncing specified folders:

```
$ portal channel <Portal-Name>
```

Synchronization of files over the channels is done continuously using `rsync`. Data transfer happens every time a new file appears or an existing file is changed in the source folder.

To stop synchronization press `^C`.

Index

Symbols

-S SNAPSHOT, –snapshot SNAPSHOT
    command line option, 12
-a, –all
    command line option, 13
-c CONFIG, –config CONFIG
    command line option, 12
-f FIELD, –field FIELD
    command line option, 14
-f, –force
    command line option, 13
-n NAME, –name NAME
    command line option, 12, 13
-s SIZE, –size SIZE
    command line option, 12, 13
-t key:value [key:value ...], –tags key:value [key:value ...]
    command line option, 12, 13
-t [session], –tmux [session]
    command line option, 14
-z ZONE, –zone ZONE
    command line option, 12

C

command line option
    -S SNAPSHOT, –snapshot SNAPSHOT, 12
    -a, –all, 13
    -c CONFIG, –config CONFIG, 12
    -f FIELD, –field FIELD, 14
    -f, –force, 13
    -n NAME, –name NAME, 12, 13
    -s SIZE, –size SIZE, 12, 13
    -t key:value [key:value ...], –tags key:value [key:value ...], 12, 13
    -t [session], –tmux [session], 14
    -z ZONE, –zone ZONE, 12