Lab - Managing Resources with Tagging

This lab is divided into two parts:

- In the **Task** portion of this lab, you will use the AWS Command Line Interface (CLI) to inspect the tags assigned to a number of Amazon EC2 instances. You will then use pre-provided scripts to shut down and start up a number of Amazon EC2 instances simultaneously, based on their tags.
- In the **Challenge** portion of this lab, you will be challenged to think of a way to terminate instances that fail to implement specific tags.

Objectives
After completing this lab, you will be able to:

- Apply tags to existing AWS resources.
- Find resources based on tags.
- Use the AWS CLI or AWS SDK for PHP to stop and terminate Amazon EC2 instances based on certain attributes of the resource.

Duration

This lab will require approximately **45 minutes** to complete.

Scenario

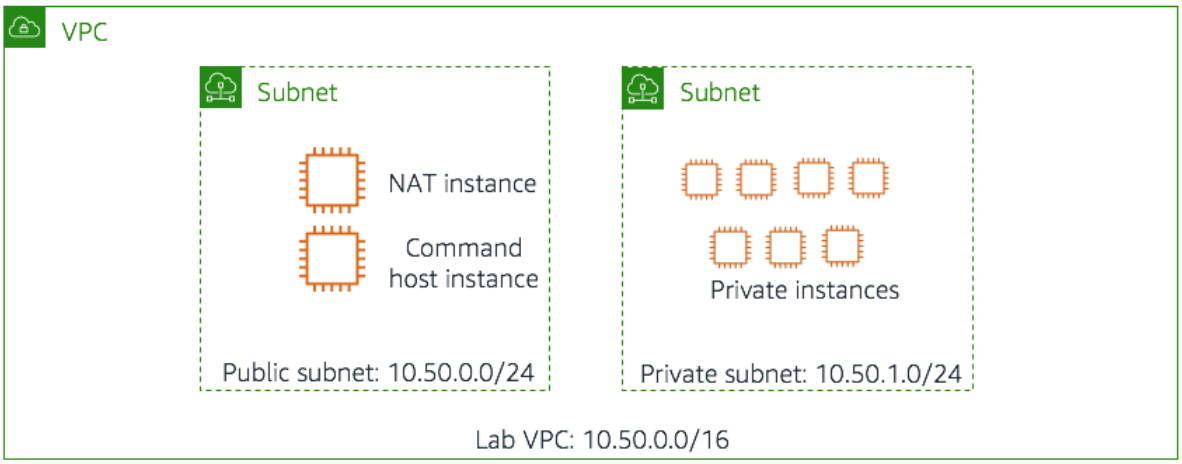
The environment for this lab (pictured below) consists of:

- Amazon VPC named Lab VPC
- Public subnet
- Private subnet
- Amazon EC2 Linux instance named CommandHost [AWS Command Line Interface (CLI) tools have been pre-installed and configured for you on this instance]
- 8 Amazon EC2 Linux instances
- Private instances have three custom tags applied to them:

Tag Name	content
Project	The project that the instance belongs to. The instances in this lab belong to one of two projects: **ERPSystem** and **Experiment1**.
Version	The version of the project that this instance belongs to. All Version tags are currently set to 1.0.
Environment	One of three values: **development**, **staging**, or **production**.

In the Task portion of this lab, you will log in to the Command Host and run some commands to find and change the Version tag on all development instances. You will run several examples that show how you can use the JMESPath syntax supported by the AWS CLI `--query` option to return richly formatted

output. You will then use a set of pre-provided scripts to stop and re-start all instances that are tagged as belonging to the **development** environment.



Accessing the AWS Management Console

1. At the top of these instructions, click Start Lab to launch your lab.

   A Start Lab panel opens displaying the lab status.

2. Wait until you see the message "**Lab status: ready**", then click the **X** to close the Start Lab panel.

3. At the top of these instructions, click AWS

   This will open the AWS Management Console in a new browser tab. The system will automatically log you in.

   Tip: If a new browser tab does not open, there will typically be a banner or icon at the top of your browser indicating that your browser is preventing the site from opening pop-up windows. Click on the banner or icon and choose "Allow pop ups."

4. Arrange the AWS Management Console tab so that it displays along side these instructions. Ideally, you will be able to see both browser tabs at the same time, to make it easier to follow the lab steps.

   ⚠ Please do not change the Region during this lab.

Task 1: Using Tags to Manage Resources

In this task, you will log in to the Command Host, and use the AWS CLI to find a set of resources according to their tags. You will then use the AWS CLI to change the value of one of the tags.

Connect to the Command Host

The following instructions now vary slightly depending on whether you are using Windows or Mac/Linux.

Windows Users: Using SSH to Connect

These instructions are for Windows users only.

If you are using macOS or Linux, skip to the next section.

5. In the **AWS Management Console**, on the **Services** menu, click **EC2**.
6. In the left navigation pane, click **Instances**.
7. Select the **Command Host**.
8. Copy the **IPv4 Public IP** from the Description in the lower pane.
9. Read through the three bullet points in this step before you start to complete the actions, because you will not be able see these instructions when the Details panel is open.
    - Click on the Details drop down menu above these instructions you are currently reading, and then click Show. A Credentials window will open.
    - Click on the **Download PPK** button and save the **labsuser.ppk** file. Typically your browser will save it to the Downloads directory.
    - Then exit the Details panel by clicking on the **X**.
10. Download needed software.
    - You will use **PuTTY** to SSH to Amazon EC2 instances. If you do not have PuTTY installed on your computer, download it here.
11. Open **putty.exe**
12. Configure PuTTY to not timeout:
    - Click **Connection**
    - Set **Seconds between keepalives** to `30`

    This allows you to keep the PuTTY session open for a longer period of time.
13. Configure your PuTTY session:
    - Click **Session**
    - **Host Name (or IP address):** Paste the **Public IPv4** value you copied to your clipboard earlier in the lab.
    - Back in PuTTy, in the **Connection** list, expand ⊞ **SSH**
    - Click **Auth** (don't expand it)
    - Click **Browse**
    - Browse to and select the lab#.ppk file that you downloaded
    - Click **Open** to select it
    - Click **Open**
14. Click **Yes**, to trust the host and connect to it.
15. When prompted **login as**, enter: `ec2-user`

    This will connect you to the EC2 instance.
16. Windows Users: Click here to skip ahead to the next task.

Mac and Linux Users

These instructions are for Mac/Linux users only. If you are a Windows user, skip ahead to the next task.

17. In the **AWS Management Console**, on the **Services** menu, click **EC2**.
18. In the left navigation pane, click **Instances**.
19. Select the **Command Host**.
20. Copy the **IPv4 Public IP** from the Description in the lower pane.
21. Read through the three bullet points in this step before you start to complete the actions, because you will not be able see these instructions when the Details panel is open.
    - Click on the Details drop down menu above these instructions you are currently reading, and then click Show. A Credentials window will open.
    - Click on the **Download PEM** button and save the **labsuser.pem** file.
    - Then exit the Details panel by clicking on the **X**.
22. Open a terminal window, and change directory `cd` to the directory where the labsuser.pem file was downloaded.

    For example, run this command, if it was saved to your Downloads directory:

    ```
    cd ~/Downloads
    ```

23. Change the permissions on the key to be read only, by running this command:

    ```
    chmod 400 labsuser.pem
    ```

24. Return to the terminal window and run this command (replace **<public-ip>** with the **Public IPv4** value you copied to your clipboard earlier in the lab):

    ```
    ssh -i labsuser.pem ec2-user@<public-ip>
    ```

25. Type `yes` when prompted to allow a first connection to this remote SSH server.

    Because you are using a key pair for authentication, you will not be prompted for a password.

Finding Development Instances For The Project

Now that you are logged in, you can use the AWS CLI to find the resources in your private subnet that belong to the **ERPSystem** project and are in the Environment named **development**. You will also see how to use the AWS CLI **--query** option to produce richly formatted results.

26. To find all instances in your account that are tagged with a tag of **Project** and a value of **ERPSystem**, copy the following command and run it in the Linux terminal window:

    ```
    aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
    ```

    The command should output the full set of parameters available for all seven instances that are tagged **Project=ERPSystem**. This is a lot of output, and most of it does not apply to this lab. In

the next step, you will use the **--query** parameter to narrow down the results.

27. Use the `--query` parameter to limit the output of the previous command to only the instance ID of the discovered instance:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
--query 'Reservations[*].Instances[*].InstanceId'
```

Your output entries will now consist of a list of instance IDs:

```
[
[
"i-135b491e"
],
[
"i-3e584a33"
],
…
]
```

The **--query** command used in this example uses the JMESPath wildcard syntax to specify that the command should iterate through all reservations and all instances and return the InstanceId for each instance in the return results.
This is an improvement over returning every property of our instances. But what if you want to include multiple fields in the output?

28. Copy the following command and run it in the Linux terminal window to include both the instance ID and the Availability Zone of each instance in your return result:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
--query 'Reservations[*].Instances[*].
{ID:InstanceId,AZ:Placement.AvailabilityZone}'
```

Two name/value pairs are returned for each result.
This command builds on the previous command's use of the JMESPath syntax by using curly braces to specify a query for multiple properties on each instance returned:

`object.{Alias1:PropertyName1,Alias2:PropertyName2,[…]}`

As seen here, you can specify an alias for each property in order to return a more abbreviated output format.

With this output, you can clearly see that your filter worked, and you are only seeing instances that are associated with the project **ERPSystem**. However, you still will probably not be able to identify which instances are being returned, based on this information. In the next steps, you will see how to include the value of your custom tags in the return output.

29. To include the value of the **Project** tag in your output, copy and run the following command in the Linux terminal:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
--query 'Reservations[*].Instances[*].
{ID:InstanceId,AZ:Placement.AvailabilityZone,Project:Tags[?Key==`Project`] |
[0].Value}'
```

Your output now includes the value of the Project tag:

```
[[{
"Project": "ERPSystem",
"AZ": "us-west-2a",
"ID": "i-3250b838"
}],
…
]
```

The value of a specific named tag can be retrieved via a JMESPath query, using the following syntax:

```
Tags[?Key==\`Project\`] | [0].Value
```

This syntax instructs JMESPath to find all elements within the **Tags** array that have a **Key** value of **Project**. The output of that command—which will be a single Tags element—is then piped to another command that selects the first instance of this filtered set and selects the named parameter **Value**, which is the value of the **Project** tag. This result is then assigned the alias **Project**.

30. Copy and run the following command to also include the Environment and Version tags in your output:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
--query 'Reservations[*].Instances[*].
{ID:InstanceId,AZ:Placement.AvailabilityZone,Project:Tags[?Key==`Project`] |
[0].Value,Environment:Tags[?Key==`Environment`] |
[0].Value,Version:Tags[?Key==`Version`] | [0].Value}'
```

The results will give you a fuller picture of the instances currently associated with the project named **ERPSystem**:

```
[[{
"Environment": "production",
"Project": "ERPSystem",
"Version": "1.0",
"AZ": "us-west-2a",
"ID": "i-3250b838"
}],
…
]
```

31. Finally, add a second tag filter to see only the instances associated with the project named **ERPSystem** that belong to the Environment named **development**:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
"Name=tag:Environment,Values=development" --query
'Reservations[*].Instances[*].
{ID:InstanceId,AZ:Placement.AvailabilityZone,Project:Tags[?Key==`Project`] |
[0].Value,Environment:Tags[?Key==`Environment`] |
[0].Value,Version:Tags[?Key==`Version`] | [0].Value}'
```

You should see only two instances returned by this command, both with a **Project** tag value of ERPSystem and an **Environment** tag value of development:


```
[[{
"Environment": "development",
"Project": "ERPSystem",
"Version": "1.0",
"AZ": "us-west-2a",
"ID": "i-9552ba9f"
}],
…
]
```


Changing Version Tag for Development Process

In this procedure, you will change all of the **Version** tags on the instances marked as **development** for the project **ERPSystem**.

You could individually set these properties on each affected instance, but an automated approach is more practical. You can use a simple Linux Bash shell script to build on the queries you built earlier and modify tag entries as a batch operation.

32. On the CommandHost, open the file **/home/ec2-user/change-resource-tags.sh**:

```
nano change-resource-tags.sh
```

33. Examine the contents of the script:

```
#!/bin/bash

ids=$(aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
"Name=tag:Environment,Values=development" --query
'Reservations[*].Instances[*].InstanceId' --output text)

aws ec2 create-tags --resources $ids --tags 'Key=Version,Value=1.1'
```

This script first uses the command `aws ec2 describe-instances` to return only a list of instance IDs for the development machines that belong to the **ERPSystem** project. It then passes those values to the `aws ec2 create-tags` command, which either creates a new tag or (in this case) overwrites an existing tag.

Notice how the first command uses the **--output text** option to manipulate the return results as text instead of as JSON. Using this command instead of JSON on a simple return result—in this case, a list of IDs—can make it easier to manipulate the return result and pass it to other commands.

34. Close the nano editor and run this command from the Linux command prompt:

```
./change-resource-tags.sh
```

35. To verify that the version number on these instances has been incremented and that other non-development boxes in the **ERPSystem** project have been unaffected, copy and run the following command:

```
aws ec2 describe-instances --filter "Name=tag:Project,Values=ERPSystem"
--query 'Reservations[*].Instances[*].{ID:InstanceId,
AZ:Placement.AvailabilityZone, Project:Tags[?Key==`Project`]
|[0].Value,Environment:Tags[?Key==`Environment`] |
[0].Value,Version:Tags[?Key==`Version`] | [0].Value}'
```

Task 2: Stop and Start Resources by Tag

In this task, you will use a pre-provided script to stop and start a set of instances tagged as development instances.

Examining the Stopinator Script

36. On the Command Host Instance, cd into the directory aws-tools in the home directory:

```
cd aws-tools
```

37. Open the file **stopinator.php** and examine its contents:

```
nano stopinator.php
```

The **stopinator.php** script is a simple script that uses the AWS SDK for PHP to stop and restart instances based on a set of tags. This enables scenarios such as shutting off your development environment servers at the end of the day and restarting them the next morning. The script will look in every AWS region for instances that match the specified tags.
The script takes the following arguments:

- **-t**: A set of tags in the following format: `name=value;name=value`
  The script converts these tags into the format expected by the AWS PHP call `Ec2::DescribeInstance()`. If this optional parameter is absent, the script will identify and shut down all running Amazon EC2 instances in the account.
- **-s**: A Boolean parameter; no arguments are required. When this parameter is present, instances identified by **-t** are started instead of stopped.

38. Exit your nano editor.

Stopping and Restarting ERPProject Development Process

In this task, you will use the stopinator.php script to bring down and bring back up your development environment for the **ERPSystem** project.

39. From the Linux shell, run the stopinator.php script:

```
./stopinator.php -t"Project=ERPSystem;Environment=development"
```

The output should look like this, indicating that two instances will be stopped in your current AWS region. (Your results will differ depending on the region in which your lab is running.)

```
Region is us-east-1
  No instances to stop in region
Region is us-west-1
  No instances to stop in region
Region is us-west-2
  Identified instance i-9552ba9f
  Identified instance i-d35fb7d9
Stopping all identified instances...
[…]
  No instances to stop in region
Region is sa-east-1
  No instances to stop in region
```

40. On the **Services** menu, click **EC2**.
41. In the navigation pane, click **Instances**.
42. Verify that two instances are stopping or have already been stopped.
43. Return to the SSH session for Command Host, and from the Linux prompt, restart your instances with the following command:

```
./stopinator.php -t"Project=ERPSystem;Environment=development" -s
```

44. Return to the EC2 Management Console window and verify that the two instances that were previously shut down are now restarting.

Task 3: Challenge: Terminate Non-Compliant Instances

In this Challenge, you will be asked to find a way to terminate instances that do not conform to certain security guidelines.

Note If you are already familiar with AWS, we recommend that you try this challenge yourself **before** reading the detailed solution provided in the next section. When you have completed the challenge, check your work by reviewing the detailed solution.

Challenge Description
Scenario: Your company wants you to create automated processes that will automatically terminate instances that might allow a possible security breach. You have identified a list of security risks and are now deciding how to implement them efficiently by using either AWS CLI commands or the PHP SDK for AWS.
Challenge: Your first security task is simple: find all instances in your private subnet that do not implement the **Environment** tag, and terminate them (i.e., a “tag-or-terminate” policy).

Hints:

- If you are not comfortable with PHP or a similar programming language (such as Python or Ruby) for which an AWS SDK is available, try to use a series of AWS CLI commands to perform this task.
- The AWS PHP call `Ec2::terminateInstances()` can terminate instances. The equivalent AWS CLI command is `aws ec2 terminate-instances`.
- You can use the **stopinator** script from section 2 as a reference for any code you write.

Challenge Solution Overview

There are multiple ways to approach this problem using a variety of programmatic or command-line solutions. The general solution to the problem consists of the following steps:

- Identify all of the instances that currently have the **Environment** tag defined.
- Compare this against the list of all available instances, and record the instance IDs of any instances that are not part of the list obtained from Task 1.
- Supply the instance IDs of the non-tagged instances to AWS by using the aws ec2 stop-instance command (AWS CLI) or the **Ec2::terminateInstances()** API call (PHP).

The following solution demonstrates how this problem could be solved using a PHP script.

Task 3.1: Review the Tag-Or-Terminate Script

45. Open the file **terminate-instances.php** with the nano editor.

    ```
    nano terminate-instances.php
    ```

46. Examine the **params** block for this script. Note that it takes two arguments: the current region that you are running in (**region**), and the ID of a subnet (**subnetid**). The code uses the **subnetid** argument to determine where to look for non-compliant instances.
47. Examine the first block of code, beginning with the comment `# Obtain a list of all`

`instances with the Environment tag set`.

This block of code uses the **describeInstances()** method, a filter to find all instances that have the **Environment** tag defined, regardless of the tag's value. It stores all of the instance IDs that it finds in a hash table.

48. Examine the second block of code.

    This code examines all instances within your subnet and compares them to the list of instances that are tagged with the **Environment** tag. If an instance is not in the tagged list, then its instance ID is added to a list of instances to terminate.

49. Examine the last section of the script.

    These lines use the list of non-compliant instance IDs as an argument to the **terminateInstances** method.

Configuring Environment to Test Script

Before running the script, you will need to alter a couple of instances in your lab so that they no longer have the **Environment** tag defined.

50. Return to your EC2 Management Console and observe the instances running in your lab environment.
51. Select one of the instances in your private subnet.
52. On the **Tags** tab for the instance, click **Add/Edit Tags**.
53. Find the Environment tag, and click the **remove** icon.
54. Click **Save**. Repeat this process for one other instance in your private subnet.

Run the Script

55. In the EC2 Management Console, select one of the instances in your private subnet.
56. On the **Description** tab for your instance, find the **Availability zone** field, and copy all but the last letter to a text file. This value will be referred to as region in a subsequent procedure.
57. Find the **Subnet ID** field, and copy its value to a text file. This value will be referred to as subnet-id in a subsequent procedure.
58. Return to your SSH session, and run the **terminate-instances.php** script (replacing the <region> with your region and <subnet-id> with your subnet-id):

    ```
    ./terminate-instances.php -region <region> -subnetid <subnet-id>
    ```

    You should see something similar to the following results:

    ```
    Checking i-dd3a90d1
    Checking i-a4248ea8
    Checking i-793a9075
    ```

```
Checking i-a9248ea5
Checking i-aa248ea6
Checking i-da3a90d6
Checking i-a13b91ad
Checking i-a23b91ae
Checking i-ab248ea7
Terminating instances...
Instances terminated.
```

Lab Complete

Congratulations! You have completed the lab.

59. Click End Lab at the top of this page and then click **Yes** to confirm that you want to end the lab.

    A panel will appear, indicating that "DELETE has been initiated... You may close this message box now."

60. Click the **X** in the top right corner to close the panel.