



Microsoft DevOps on AWS

Ninja Bootcamp

Lab Guide

v1.0 – R.1.3

Contents

110 – CI/CD introduction (Optional)	1
120 - Lab setup.....	2
210 - Azure DevOps server crash course (Optional)	3
220 – Add a new agent to Azure DevOps server (Optional)	4
300 – Integrate AWS PowerShell with Azure DevOps/TFS	5
400 – Integrate AWS CloudFormation with Azure DevOps/TFS	8
500 – Integrate AWS CodeDeploy with Azure DevOps/TFS.....	11
600 – Integrate AWS Elastic Beanstalk with Azure DevOps/TFS	14
700 – Integrate AWS Fargate with Azure DevOps/TFS	16
List of videos	18

Timing Tips

- **Plan your time carefully, you can easily run out of time.**
- Skip optional labs (mandatory=**No**) and do them later if you have spare time.
- First scan through the questions you need to answer and then start watching the relevant video. To save the time, try to do the labs while watching the videos.
- You can watch the videos at a higher speed (1.5x)/manually fast forward
- Execute labs starting with 1xx/2xx/3xx as quickly as possible. Do not spend too much time watching videos. Try to reach 400+ topics as soon as possible.
- Once you complete 4xx topics, you can start 500,600,700 in any order. For example, if you find container topics interesting, after completing 440, jump straight into 700.
- **Spend more time doing hands-on activities**

Plan your time carefully, spend more time doing hands-on activities instead of watching videos

Id	Title	Mandatory	Instruction (min)	Estimated* time to complete the lab (min)	
			Video time	Average user	Expert user
	100 - CI/CD introduction	No	13	13	5
	120 - Lab setup	Yes	14	20	15
	200 – Azure DevOps basics	No	57	90	60
	210 - Azure DevOps server/TFS - crash course	No	38	60	40
	220 - Azure DevOps server/TFS – add a new build agent	No	19	30	20
	300 - Integrate AWS PowerShell with Azure DevOps/TFS	Yes	56	110	40
	310 - AWS PowerShell introduction & lab setup	Yes	11	15	10
	320 - Basic credential management	Yes	5	15	5
	330 - Using profiles to keep credentials	No	5	15	5
	340 - Using EC2 instance profiles to grant permission	Yes	20	35	5
	350 - Use temporary credentials with STS assume role	No	15	30	15
	400 - Integrate AWS CloudFormation with Azure DevOps/TFS	Yes	85	139	60
	410 - AWS CloudFormation introduction	No	11	11	5
	420 - Manually deploy a CloudFormation template	No	6	10	5
	430 - Command line deployment of CloudFormation template	Yes	18	28	15
	440 - Integrate CloudFormation with Azure DevOps using extensions	Yes	36	60	25
	450 - Integrating CloudFormation with Azure DevOps using PowerShell	No	14	30	10
	500 – Integrate AWS CodeDeploy with Azure DevOps/TFS	Yes	98	202	125
	510 - AWS CodeDeploy Introduction	Yes	4	4	5
	520 - Lab introduction	Yes	10	15	15
	530 - Build deployment archive	Yes	13	23	15
	540 - Integrate Azure DevOps/TFS pipeline with S3	Yes	8	20	10
	550 - Setup AWS CodeDeploy application and deployment groups	Yes	20	35	25
	560 - Build full end to end pipeline	Yes	13	25	10
	570 – Hybrid deployments	No	30	80	45
	600 – Integrate AWS Elastic Beanstalk with Azure DevOps/TFS	Yes	51	122	82
	610 – Elastic Beanstalk introduction	Yes	5	5	5
	620 – Lab introduction	Yes	7	15	10
	630 - Build a deployment archive & upload to S3	Yes	10	20	15
	640 – Create an AWS Elastic Beanstalk application	Yes	11	20	15
	650 – Elastic Beanstalk under the hood	No	7	7	7
	660 – Finishing the full pipeline	Yes	11	25	15
	670 – Advanced Elastic Beanstalk scripting with PowerShell	No	0	30	15
	700 – Integrate AWS Fargate with Azure DevOps/TFS	Yes	88	163	100
	710 – Container introduction	No	9	9	10
	715 – Crash course on AWS container orchestration technologies	Yes	9	9	10
	720 – Lab Setup: Build the ASP.NET web app container image on build agent	Yes	13	20	15
	730 – Create an Amazon ECR repository and publish the container image	Yes	11	25	15
	740 – Create AWS Fargate Task	Yes	6	20	10
	750 – Create AWS Fargate Cluster	Yes	7	20	5
	760 – Create AWS Fargate Service	Yes	14	25	15
	770 – Integrate AWS Fargate with the full pipeline.	Yes	19	35	20
Total – watch all videos and execute all labs (in hours)				15	8.5
Total – watch only mandatory videos and execute only mandatory labs (in hours)				9	4
Total – only watch mandatory videos and execute mandatory labs & only watch the videos related to optional labs (in hours)				12	6
Total – watch all videos and do no labs (in hours)				8	6
Total – watch only mandatory videos and do no labs (in hours)				5	4

110 – CI/CD introduction (Optional)

If you are not familiar with CI/CD (Continuous Integration/Continuous Delivery) systems, quickly go through following crash course to get yourself comfortable with the software build lifecycle and learn the purpose of a CI/CD system. **If you are already familiar with CI/CD systems (E.g. you are a developer), we strongly recommend you skip this introduction.**

Watch the video: <https://youtu.be/rwM4nQ0hYxl>



120 - Lab setup (Mandatory)

Learning objectives

You will get yourself familiarized with the lab environment and start the lab machines

Lab instructions

Watch the video: <https://youtu.be/AbMeXc9t7JI>

Then do the following exercises

E 1. Download the zip file

<https://s3-ap-southeast-2.amazonaws.com/aws-ninja-bootcamp/ms-devops-on-aws/assets.zip>

to your lab machine/laptop and unzip it to a folder called `<assets>`. Throughout this lab series we will use the folder name `<assets>` to refer to this folder.

Your instructor has given you a temporary AWS account to run this lab series. Use those credentials and login to the AWS account. **Make sure you login using the URL specified to the specified AWS region.** You do not have access to regions other than what is given in your account handover email. You can only use the services specified in the lab guide.

As shown in the video, start the **DevMachine** and **BuildMachine** and login into them. Remember when logging into these machines using an RDP connection, use the prefix `.\` (Example `.\dev-user`)

	Image Name x is the version	Instance Size	Username	Password
DevMachine	DevOps-DevMachine-Vx	t2.large	dev-user	ILoveAWS@@##998877\$\$
BuildMachine	DevOps-BuildMachine-Vx <i>Note: Do not use DevOps-BuildMachineLinux-Vx, we will use it later</i>	t2.medium	build-user	ILoveAWS@@##887766\$\$

Remember:

- You have permissions to start following instance types. If you try to start any other instance type, you will get permission denied error [t2.small , t2.medium, t2.large, t3:small, t3.medium, t3.large].
- When giving names for new AWS resources, please use the same names as shown in the videos. This will make your life easy when navigating the instructions.

E 2. As described in the video, test whether the BuildMachine can communicate with the DevMachine.

E 3. As shown in the video, from the DevMachine, navigate to <https://dev/DefaultCollection/> and view Azure DevOps server UI. Your **first** page loading can be slow due to cache refresh, please be patient.



210 - Azure DevOps server crash course (Optional)

If you are already familiar with Azure DevOps/Visual Studio Team Services (VSTS) or latest versions of Team Foundation Server (TFS), this lab is optional. Skip this if possible and come back again if you have spare time.

If you have not used Azure DevOps/Visual Studio Team Services (VSTS) or latest version of Team Foundation Server (TFS) before please **quickly** go through this lab.

Learning objectives

In this lab you will learn basics of Azure DevOps. You will...

- Learn Azure DevOps server project structure
- Learn Azure DevOps server agent/agent pools
- Learn how to create a pipeline, add pipeline steps to the pipeline and execute it
- Learn how to add conditions/demands and automatically select Azure DevOps agent based on requirements
- Explore Azure DevOps market place and AWS extensions for Azure DevOps
- Learn built-in/predefined variables such as \$(Build.SourcesDirectory) and how to use them to identify dynamic/build time parameters.

<https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables>

Lab instructions

You will create a simple pipeline and execute it on an agent. You can skip this if you are already familiar with Azure DevOps server.

Watch the video: <https://youtu.be/7NPw0MZ1kuw>

Then do the following exercises

- E 1. [Video timestamp: 0min-18min]: Create a simple Azure DevOps pipeline under the project BuildTest. Add two build steps, one command line step that prints the message “hello from command line 123” and another PowerShell script that prints the message “hello from PowerShell 123” to the console.
- E 2. [Video timestamp: 18min-21min]: As shown in the video, disable the agent pool and queue a new pipeline execution. Observe what happens to the execution. Enable the agent pool again.
- E 3. [Video timestamp: 21min-31min]: Add a condition/demand that requires AWSPowerShell tools to be installed on the build agent by demanding AWSTools equals “AWSPowerShell”. Manually tag this capability in the build agent and execute the pipeline again.
- E 4. [Video timestamp: 31min-37min]: Modify the PowerShell step defined in E 1 to save the message “hello from command line 123” to a file called “myfile.txt” inside **<the current staging folder of the current build>** \myfolder\<myfile.txt>. Use built-in variables to identify the **<current staging folder of the current build>**, and a custom variable called myfilename to identify **<myfile.txt>**.
You may find following PowerShell commands useful.

New-Item

Out-File

Predefined variables for Azure DevOps are given at

<https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables>



220 – Add a new agent to Azure DevOps server (Optional)

This lab is optional. We **strongly recommend** you skip this lab and come back again if you have spare time.

Learning objectives

In this lab, you will learn how to add a new agent to Azure DevOps server. You can run your build agent in any other cloud provider, on-premises or in AWS. For example, you can use Azure DevOps services in Azure, and add a build agent in AWS to do AWS specific deployments.

Lab instructions

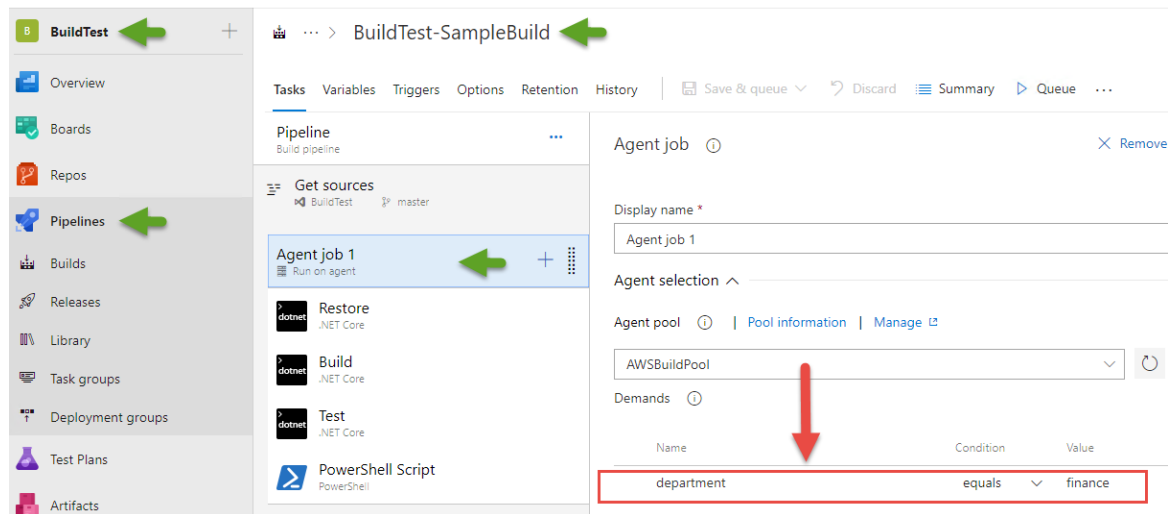
Watch the video: <https://youtu.be/lkB7HTbdmhA>

Then do the following exercise

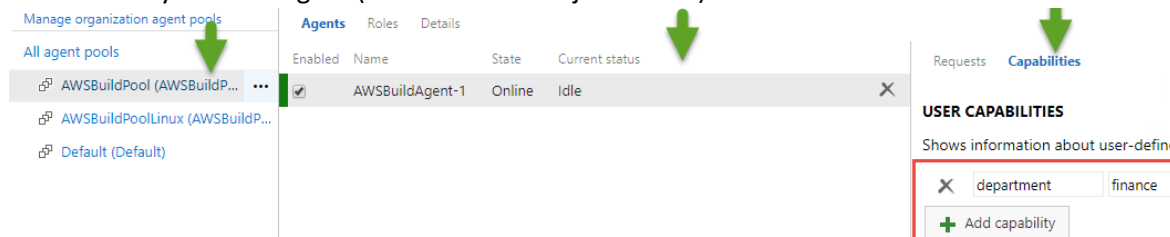
- E 1. [Video timestamp:0min-11min]: As shown in the video, add a new build agent
- E 2. [Video timestamp:11min-16mi]: As shown in the video, fix any errors with certificates, and get Build-Test project to build on newly added agent
- E 3. [Video timestamp:16min 45sec-18min]: As shown in the video, remove the newly added agent.

Debugging Tips:

Agent jobs in project pipelines can demand certain capabilities from build agents. For example, BuildTest project requires the agent to have a custom property of “department” with a value of “finance”.



You can manually define these capabilities by adding a key value pair. If that is not in place, when attempting to run a build the only enabled agent (the one that was just added) will not execute the build.



300 – Integrate AWS PowerShell with Azure DevOps/TFS

Learning objectives

You will learn basics of AWS PowerShell commands and learn different methods to provide credentials to AWS PowerShell. AWS PowerShell is heavily used when integrating CI/CD systems with AWS. Our focus is not to learn all the AWS PowerShell cmdlets, but to learn the fundamentals that allows you to apply the knowledge for any cmdlet. Objective here is not be a PowerShell master, we **strongly recommend you to spend not more than 40min on this lab**.

Lab instructions

E 1. 310: AWS Tools for PowerShell introduction: Watch the video: <https://youtu.be/CjF2wHR5-XE>

[Video timestamp:0min-5min 45sec]: You learned...

- Credential search order
- Where to find help for AWS PowerShell

[Video timestamp:9min-11min]: Now create an S3 bucket and try to list the buckets

E 2. 320: Basic credential provisioning: Watch the video: <https://youtu.be/KENymwb4EFc>

- [Video timestamp:0min-1min 30sec]: As shown in the video, create an IAM user called `mytest-user` and note down it's Access Key and Secret Key

Important: You need to use the same name `mytest-user`. If you are executing this lab in an AWS provided locked-down lab accounts, you will not have permission to create a user with a different name.

- [Video timestamp:1min 30sec-3min 30sec]: As shown in the video, use IAM user credentials directly in the command itself.
Why it's not recommended to use IAM credentials in this way? How these credentials can leak?
- [Video timestamp: 3min 30sec-4min 40sec] Use IAM user credentials in the PowerShell session. What happen to those credentials, when you close the PowerShell window?

330: Credential profiles (**Optional lab**, we recommend to skip this lab and comeback again if you have spare time):

Watch the video: <https://youtu.be/FUnRmGi4xiw>

- As given in the video, create a profile in the default location to keep the access key and the secret key. Can someone other than the user who creates it access those credentials?
- How to get a list of profiles saved in the default profile location?
As given in the video, use those credentials stored in a profile to list S3 buckets.
- As given in the video, create a profile, but this time keep it under a given file name. Use the credential in the file to list S3 buckets.
Can you keep these credential files in a network shared location?
- How many profiles can you keep per user per machine?
- Think how you can you use different profiles (with just enough permission granted) to do different deployment tasks
- Remove the profiles you have created



E 3. 340: EC2 instance profiles (**Important Lab**): Watch the video: <https://youtu.be/BOUj7gzjFpo>

- Why it is not recommended to keep access key and secret key in the instance as a profile/file?
Tip: Think along the lines of rotating these keys in the future, what will happen if the instance is compromise?
- What is the difference between removing the access key and making it inactive? How many access keys you can have per IAM user?
- Can you guess the advantage of having multiple access keys for a given user during key rotation?
- What AWS services allows you to run security validation checks to verify the age of an access key?
- As shown in the video, make the access key inactive, active, inactive and delete. Finally delete the IAM user `mytest-user`. In each step, check whether you can list S3 buckets with old access keys.
- As shown in the video, modify the EC2 instance profile and attach the managed permission `AmazonS3FullAccess` to the EC2 instance's IAM role. Now try to list S3 buckets and check whether you can successfully execute the cmdlet. *Note that it can take a few seconds to get the changes reflected. If you execute the cmdlet immediately after attaching the permission, you may get Access denied error because the token update takes a few seconds.*

E 4. 350: Use temporary credentials with Secure Token Service (STS) (Optional): Watch the video:

<https://youtu.be/EJqovjmHL5w>

- [Video timestamp: 0min-3min]: As shown in the video, remove `AmazonS3FullAccess` that we attached in E4. Now create a new user called `mytest-user` with no permissions and create access key and secret key. Make sure you use the same username as `mytest-user` since you don't have permission to create a user with a different name.
- [Video timestamp: 3min-6min]: As shown in the video, create a new role called `my-test-role` and attached the permission `AmazonS3FullAccess` to it. Modify the trust section of the role to trust the user `mytest-user` (we created a moment ago) to assume the role.

Note that the AWS **account number** will be different for your account.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789123:user/mytest-user"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Can we configure the trust relationship such that a user in a shared service account/pre-production account is trusted to assume a role in production environment to do production deployments?

Can a role be configured/trusted to assume another role in another account?



- [Video timestamp: 6min-8min 30sec:] As shown in the video, use the cmdlet `Use-STSRole` to assume the role `my-test-role` and get temporary credentials.
- [Video timestamp: 8min 30sec-10min 10sec] As shown in the video, use the temporary credentials you got from `Use-STSRole` to list S3 buckets. Observe how the credential changes every time you call `Use-STSRole` cmdlet. Check the expiration time of the credentials you got.
- [Video timestamp: 10min 10sec-13min] What's the minimum duration you can assume a role? Why do you think AWS has selected a minimum duration as such? (Tip: Think about large distributed systems). Try to use the credentials after they are expired and check the error message you get.
- [Video timestamp: 13min-15min] Learn how assume role method can be used to do complex deployments from a shared service account to different environments.



400 – Integrate AWS CloudFormation with Azure DevOps/TFS

Learning objectives

In this lab, you will learn different ways you can integrate AWS CloudFormation with Azure DevOps. You can use command line/PowerShell based approach to integrate AWS CloudFormation with other build systems like Jenkins, Bamboo and TeamCity.

Lab instructions

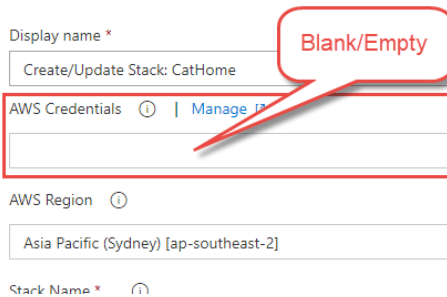
- E 1. 410: AWS CloudFormation Introduction (Optional): Watch the video: <https://youtu.be/-f8Rg-FCelc>
If you are not familiar with AWS CloudFormation watch the video to quickly learn basics of AWS CloudFormation. If you are already familiar with AWS CloudFormation, you can safely skip this introduction.
- E 2. 420: Manually deploy an AWS CloudFormation template:(Optional): watch the video: https://youtu.be/0xl_fDoy3yY
If you are already familiar with deploying CloudFormation templates, you can safely ignore this step.
- Manually deploy the given CloudFormation template. You will find the CloudFormation template at `<assets>\420_cloudformation_manual_deploy\network.template`
 - What's the credentials, CloudFormation used to provision the environment?
 - As shown in the video, delete the CloudFormation stack and check what happen to the VPC and subnets the CloudFormation has provisioned.
- E 3. 430: CloudFormation PowerShell based deployment: watch the video: https://youtu.be/QjZgWtMR_no
- [Video timestamp:0min-10min 10sec]: As shown in the video, check what happen when you try to execute the CloudFormation deploy script without proper permission.
Now attach `MyCatApp-Policy` to your DevMachine IAM role and try to execute the script again.
 - [Video timestamp:10min 20sec-14min 10sec]: As shown in the video, add a new subnet to the template and deploy the CloudFormation template. Observe how delta changes get deployed. Now remove a subnet from the CloudFormation template and then deploy the stack again. Observe how delta changes get provisioned.
 - [Video timestamp:14min 10sec-16min 15sec]: Read `MyCatApp-Policy` carefully and then try to change the PowerShell parameter `$stackName` to something like `CatHome1`. Execute the template again and observe the access denied exception you got. Can you think of using similar permissions to isolate different environments?
- E 4. 440: Integrate AWS CloudFormation with Azure DevOps using Azure DevOps's marketplace plugins: watch the video: <https://youtu.be/IRcXpRSkBwE>
- [Video timestamp: 0min - 5min 16sec] In this lab we will focus only on `MyCatApp-Plugin-Base-Pipeline`. Therefore, as shown in the video, first disable the pipeline `MyCatApp-PowerShell-Pipeline`, enable `MyCatApp-Plugin-Base-Pipeline` and enable continuous integration for `MyCatApp-Plugin-Base-Pipeline`



- [Video timestamp: 5min 16sec-7min] Search Azure DevOps extension market place for AWS plugins. You don't need to install the plugins because AWS plugins are already installed for you.
- [Video timestamp: 7min – 12min 20sec] As shown in the video, using the pipeline `MyCatApp-Plugin-Base-Pipeline`, deploy the CloudFormation template. Verify that the IAM policy `MyCatApp-Policy` is attached to the BuildMachine's IAM role, if it's not attached you can manually attach it.

Debug Tip:

It's crucial that you don't specify any connection in the plugin's AWS Credentials (Leave it blank). This will allow the plugin to fall back into credentials associated with E2 instance profile of the build agent. If you specify a connection, that connection gets the priority. E.g.



Display name *

Create/Update Stack: CatHome

AWS Credentials ⓘ | Manage IAM

AWS Region ⓘ

Asia Pacific (Sydney) [ap-southeast-2]

Stack Name * ⓘ

- [Video timestamp: 12min 20sec - 19min 45sec] As shown in the video, using the pipeline `MyCatApp-Plugin-Base-Pipeline`, deploy the CloudFormation template. Use an IAM user's access key and secret key to create a connection to AWS and use that connection in the CloudFormation deployment task. The user has to have the permission `MyCatApp-Policy`. Make sure you detach the policy `MyCatApp-Policy` from the BuildMachine's instance profile.

If you are doing this lab using AWS provided lab environment, you only have permission to create an IAM user with the name `mytest-user`.

- [Video timestamp: 19min 45sec – 25min]: As shown in the video, create a role called, `my-test-role`, attach the policy `MyCatApp-Policy` and then configure the role to trust `mytest-user`. Remove `MyCatApp-Policy` from `mytest-user` and from build machine's instance profile. Now configure the plugin task to assume the role `my-test-role` when it does deployments. What's the advantage of using **assume role** method over other methods we discussed before?
- [Video timestamp: 25min-29min 30sec] Why did we learn different methods to provide credentials to the pipeline?
- [Video timestamp: 29min 30sec - 36min] As shown in the video, use Azure repo to maintain your CloudFormation template. Open the project `MyCatApp` on Visual Studio code, add a new subnet to the `network.template`, commit the changes and push it to Azure Repo. Make sure you have enabled continuous integration for the pipeline and observe how your CloudFormation gets deployed. Similarly, remove the newly added subnet, save the file, commit the changes to the local repository and push it to the remote repository and observe how the pipeline gets executed.
- Why it is a good practice to maintain infrastructure as code in a repository?



- Quickly scan through some CloudFormation samples at <https://aws.amazon.com/quickstart>

E 5. 450: Integrate AWS CloudFormation with Azure DevOps using PowerShell **(Optional Lab)**

Watch the video: https://youtu.be/W_P8dOQg6Gs

In this lab we are going to do the same things we have done in E4, but using PowerShell. If your build system (E.g. Jenkins, TeamCity) does not have plugins from AWS, you can use this method to integrate with AWS.

- [Video timestamp: 0min-4min] As shown in the video, disable `MyCatApp-Plugin-Base-Pipeline`, enable `MyCatApp-PowerShell-Pipeline`, and enable continuous integration for `MyCatApp-PowerShell-Pipeline`.
- As shown in the video, get `MyCatApp-PowerShell-Pipeline` to deploy your CloudFormation template. What's the advantage of using Azure DevOps variables?
- What's the solution we have provided to keep the build waiting until the CloudFormation deployment finishes? Why do you think asynchronous invocation of `Create/Update stack` is useful?
- Can you think of a method to provide temporary credentials to the PowerShell script that runs on the build agent instead of attaching permission to the build agent? Tip: we already discuss this in (350: Use temporary credentials with Secure Token Service (STS))



500 – Integrate AWS CodeDeploy with Azure DevOps/TFS

Learning Objective:

In this lab, you will learn how to integrate AWS CodeDeploy with Azure DevOps server to do software deployments into hybrid environments.

Lab Instructions

E 1. 510: Aws Code Deploy Introduction: Watch the video: <https://youtu.be/bb0IKI8FBXU>

E 2. 520: Lab Introduction: Watch the video: <https://youtu.be/yYzun7WJeJE>

As shown in the video, explore the project MyDonkeyApp, build it both from command line and from Visual Studio code and run it locally. Change the message written to the console and commit it to Azure Repo.

E 3. 530: Build deployment package: Watch the video: <https://youtu.be/FQzT9if-MoE>

- As shown in the video, get your pipeline to build the deployment archive
- Explore the content of the archive and ensure you have the correct files in the correct folder structure
- Quickly read some of the predefined variables available for you in Azure DevOps
<https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables>

E 4. 540: Upload deployment package to S3: Watch the video: <https://youtu.be/D1i7rqdMQwI>

As shown in the video, get your pipeline to upload the zip archive to an S3 bucket

E 5. 550: Create AWS CodeDeploy Application: Watch the video: <https://youtu.be/82pNkm6Op-w>

E 6. **Debugging tips:** If your build gets stuck (waiting for a long period), make sure your CodeDeploy agent service is up and running.

- [Video timestamp: 0min-6min] As shown in the video, create an AWS CodeDeploy application
- [Video timestamp: 6min-12min 30sec] As shown in the video, deploy MyDonkeyApp to DevMachine. Check whether the PowerShell scripts get executed.
- [Video timestamp: 12min 30sec – 15min] As shown in the video, verify that AWS CodeDeploy agent is running as a service in the target EC2 instance. Check the log files AWS CodeDeploy agent has generated. Think of some situations where you can use these log files to debug CodeDeploy errors.
- [Video timestamp: 15min – 19min] As shown in the video, add an additional machine (BuildMachine) to the deployment group and do a new deployment targeting two machines.

E 7. 560: Linking AWS CodeDeploy with Azure DevOps: Watch the video: <https://youtu.be/V7ILGIsC6jU>

As shown in the video, get the Azure DevOps pipeline fully integrated with AWS CodeDeploy. Set up the pipeline in a way that when a developer commits a change to the Azure Repo, the software gets delivered automatically.

If you are running this lab on AWS provided environment, you can only attach AmazonS3FullAccess & AWSCodeDeployFullAccess to BuildMachine IAM role. Alternatively, you can attach the custom permission MyDonkeyApp-Policy to the build machine.



E 8. 570: Hybrid deployment with AWS CodeDeploy: Watch the video: <https://youtu.be/wM5RD1F9z-U>
Please note that this lab requires you to install some software on your laptop/desktop at your own risk.

For Mac and Linux users

We only provide lab instructions for a Windows based on-premises instance. If you are a Linux user, you can use AWS CodeDeploy documentation at <https://docs.aws.amazon.com/codedeploy/latest/userguide/instances-on-premises.html>

Alternatively, if you are a Mac or Linux user, you can start a Windows EC2 instance and simulate an on-premises instance with it.

- [Video timestamp:0min-4min] Learn the high-level steps we will execute to setup the hybrid deployment environment
- [Video timestamp:4min-5min 18sec] As shown in the video, install AWS tools for PowerShell. You can download AWS tools for PowerShell from <https://aws.amazon.com/powershell/>
- [Video timestamp:5min 18sec-8min 20sec] As shown in the video, install AWS CodeDeploy agent. You can download the agent from following location <https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install-windows.html>
Make sure the AWS CodeDeploy agent service starts properly
- [Video timestamp:8min 20sec – 10min 6sec] As given in the video, create an IAM user called `my-onpremises-machine-user`. AWS CodeDeploy Agent will use this user's access key and secret key to download deployment packages from an S3 bucket. Attach the permission, `AmazonS3ReadOnlyAccess`. In production environments, we give fine-grained permissions that only allows access to specific bucket/folder/file path pattern. Note that you only have permission to create a user with the name `my-onpremises-machine-user`.
Note down the access key and secret key
- [Video timestamp: 10min 6sec-11min 6sec] As shown in the video, create an IAM user called `my-codedeploy-onprem-reg-user`. We will use this user's access key and secret key to register an on-premises machine. Attach the permission `AWSCodeDeployFullAccess`. Unlike in this lab environment, in production environments, we give fine-grained permissions. Once the instance is registered, you can remove this user/keys.
Note down the access key and secret key
- [Video timestamp: 11min 6sec-18min 55sec] As shown in the video, alter `conf.onpremises.yml` file, add the credentials of `my-onpremises-machine-user` and restart the AWS CodeDeploy agent. Verify the logs of AWS CodeDeploy agent to make sure it does not give any access denied errors.
- [Video timestamp: 18min 55sec-22min 35sec] As shown in the video, open PowerShell, set the credentials of `my-codedeploy-onprem-reg-user` s' credentials on PowerShell session and then register the on-premises instance/laptop using `Register-CDOOnPremiseInstance` cmdlet.
Command documentation is given at <https://docs.aws.amazon.com/powershell/latest/reference/items/Register-CDOOnPremiseInstance.html>
As shown in the video, verify that the on-premises instance is properly registered under AWS CodeDeploy.



- [Video timestamp: 22min 35sec-26min] As shown in the video, tag the on-premises instance so that we can target it using deployment groups. Create a new hybrid deployment group, and include DevMachine and your on-premises instance/laptop in this deployment group.
- [Video timestamp: 26min – 28min 30sec] As given in the video, manually execute a new deployment to deploy MyDonkeyApp to on-premises instance and EC2 based DevMachine.
- Modify the Azure DevOps pipeline in MyDonkeyApp to do an end-to-end delivery to hybrid deployment group.
- [Video timestamp: 28min 30sec - 28min 30sec] Now deregister the code deployment agent (use the PowerShell cmdlet `Unregister-CDOnPremiseInstance`), delete both `my-onpremises-machine-user` and `my-codedeploy-onprem-reg-user`. Uninstall any artefacts you installed (as part of the lab) in your laptop/on-premises instance. Reset the lab environment and any related changes you made to your laptop/on-premises instance.



600 – Integrate AWS Elastic Beanstalk with Azure DevOps/TFS

Learning Objective

In this lab, you will learn how to integrate AWS Elastic Beanstalk with Azure DevOps server. We will learn how to automate the deployment of an ASP.NET core MVC application to AWS Elastic Beanstalk.

Lab Instructions

E 1. 610: AWS Elastic Beanstalk introduction (Optional): Watch the video: <https://youtu.be/qLL4k1gim8E>

If you are not familiar with AWS Elastic Beanstalk quickly watch the video: to learn what Elastic Beanstalk is.

E 2. 620: AWS Elastic Beanstalk lab introduction: Watch the video <https://youtu.be/gS6G73tThWQ>

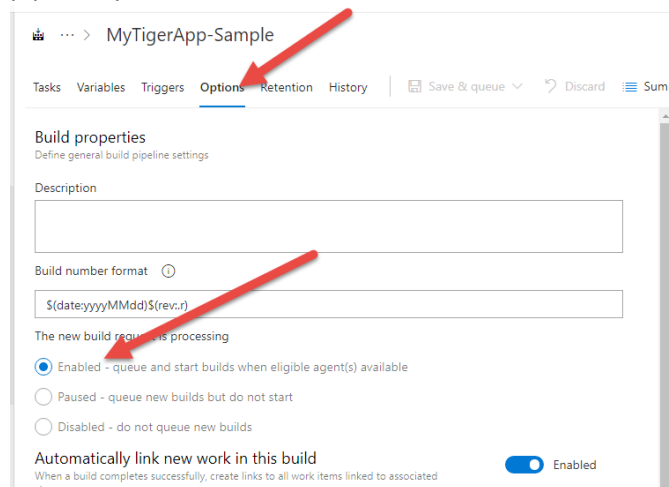
As shown in the video...

- Open MyTigerApp, build it in local machine (DevMachine) using Visual Studio code, and run it in local machine (DevMachine). (You may notice a different welcome message, than what is shown in the video)
- Make a modification to the Home page and push those changes to Azure Repo.

E 3. 630: Build a deployment archive & upload to S3: Watch the video <https://youtu.be/ga6FogFpQeY>

Debugging tip

Since there are two pipelines, make sure the pipeline you want is enabled and not paused or disabled. Disable the pipeline you don't want to execute.



As shown in the video...

- Get the pipeline to build a deployment zip package and upload it to an S3 bucket. You may need to use a different bucket name since the bucket name has to be unique for a given region
- Inspect the content of the uploaded zip file and ensure the folder structure is correct

E 4. 640: Create an AWS Elastic Beanstalk application: Watch the video: <https://youtu.be/og0mQaAr414>

As shown in the video...

- [Video timestamp: 0sec - 30sec] Create an AWS Elastic Beanstalk Application called MyTigerApp
- [Video timestamp: 30sec - 7min] Create an environment called MyTigerApp-Dev
 - Provide a key pair (so that we can remote login into the underlying instances and explore them)
 - Select the IAM instance profile as MyTigerAppEC2Role-{Id}



- Select the service role as MyTigerAppServiceRole-`{id}`
- Configure the network and use DevOps-VPC-`{id}`
- Select an EC2 instance size of t2.small and allocate about 30GB of storage

Note that in production environments, we usually don't define a key pair so that we can't remote login into the environment.

- [Video timestamp: 7min – 10min 49sec] Download the deployment zip file we uploaded to S3, and manually deploy it to MyTigerApp-Dev environment. After a few moments, check whether the website is working.

E 5. 650: Elastic Beanstalk under the hood (Optional lab): Watch the video <https://youtu.be/9sAKPYjGQ2E>

Explore under the hood operations of Elastic Beanstalk

As shown in the video...

- Explore the CloudFormation template Elastic Beanstalk has provisioned
- Explore the EC2 instance started by Elastic Beanstalk by remote login into it
- Explore how you can define your own Amazon machine image for Elastic Beanstalk
<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.customenv.html>

E 6. 660: Completing the Elastic Beanstalk deployment pipeline: Watch the video: <https://youtu.be/UfxGZjGzVcw>

As shown in the video,

- Do the final wiring for the pipeline and get the pipeline to deliver the solution to Elastic Beanstalk. Simulate a developer making a change to the home page and make sure the application gets delivered from the time the developer commits a change to Azure repo.

E 7. 670: Advanced Elastic Beanstalk deployment with PowerShell API (optional lab, but strongly recommended to do at some point)

- Your instructor has given you a sample PowerShell script at C:\Dev\MyTigerApp\ElasticBeanstalkDeployment\AdvancedDeployment.ps1 in your DevMachine. Read it carefully.
- With the knowledge you gained in previous labs, use this PowerShell script in an Azure DevOps pipeline task and deploy an Elastic Beanstalk application. You need to ensure following criteria are met
 - You do not use AWS Elastic Beanstalk deployment plugin that comes with AWS extensions for Azure DevOps
 - If the Elastic Beanstalk application and the environment do not exist, they get automatically created. You cannot assume that the Elastic Beanstalk application and environment exists.
 - Elastic Beanstalk application name and environment name are passed as variables to the script. This allows testers to spin up environments with the names they like by simply changing the build variables.

Important Tip: When you run the script, you may get an access denied error when the build agent try to create the Elastic Beanstalk environment. This is because you don't have certain permission to create roles. Look at the log files and try to identify the error.

Fix the error by properly passing the Elastic Beanstalk instance profile and service roles. Tip

Read: <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/command-options-general.html>

Now check how you can pass a value (e.g. MyTigerAppEC2Role-`{id}`) to `IamInstanceProfile` in the namespace `aws:autoscaling:launchconfiguration` and another value (e.g.

MyTigerAppServiceRole-`{id}`) to `ServiceRole` in the namespace

`aws:elasticbeanstalk:environment`



- Imagine a tester wants to have a new environment called, MyTigerApp-Test1. How can he change the build variable and execute the pipeline to create a new environment called MyTigerapp-Test1?

700 – Integrate AWS Fargate with Azure DevOps/TFS

Learning Objectives

We will learn how to create a CI/CD pipeline to deploy containers to a container cluster.

Lab Instructions

- E 1. 710: Container Introduction (Optional): If you are not familiar with container basics watch the introduction video <https://youtu.be/JJGn1dCvwWg>
- E 2. 715: Crash course on AWS container orchestration technologies: Watch the video <https://youtu.be/5EpDeiPHLFQ>
- E 3. 720: Lab Setup: Watch the video: <https://youtu.be/H9pjj20oeks>
- [Video timestamp: 0sec – 4min 50sec] As shown in the video, build MyWolfApp website in DevMachine and test it. Make a modification to the home page and commit it to Azure Repo
 - [Video timestamp: 4min 50sec – 12min 45sec] As shown in the video, start the Linux build agent so that you can build Linux containers. Once the build agent comes alive, execute the pipeline and test whether the agent can build the container image.

Although not needed for the lab, if you want to ssh into the build agent you can start the build agent with a key pair.

Username: **ubuntu**

How to SSH into the instance with a key pair is given at

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstances.html>

- E 4. 730: Publish a container image to Amazon ECR: Watch the video: <https://youtu.be/ymV-MIalGSA>

[Video timestamp: 0sec – 2min 10sec] As shown in the video, create an Amazon ECR repository

[Video timestamp: 2min 10sec-11min] Get your build agent to publish MyWolfApp container image to the Amazon ECR repository.

Note that throughout this lab series, when granting permissions, we follow an easy approach (E.g. attaching the managed permission `AmazonEC2ContainerRegistryFullAccess`). In production environments, we create fine grained permissions like the one below that gives access only to a specific ECR repository (mywolfapp).



```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AuthToken",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "ap-southeast-2"
        }
      }
    },
    {
      "Sid": "RepoOperations",
      "Effect": "Allow",
      "Action": "ecr:*",
      "Resource": "arn:aws:ecr:ap-southeast-2:123456789101:repository/mywolfapp",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "ap-southeast-2"
        }
      }
    }
  ]
}

```

E 5. 740: Create AWS Fargate task definition: Watch the video: <https://youtu.be/8FlnjhBCsYs>

As shown in the video, create an AWS Fargate task definition pointing to the container registry you have created. Make sure you point the task definition to the latest container image (with :latest tag).

E 6. 750: Create an AWS Fargate cluster and test it. Watch the video: <https://youtu.be/F85-Ok47X88>

- [Video timestamp: 0sec – 1min 30sec]: As shown in the video, create an AWS Fargate cluster.
- [Video timestamp: 1min 30sec-6min 29sec]: As shown in the video, run the task you created in E5 and test whether the cluster is working as expected.
- [Video timestamp: 6min 10sec-6min 29sec]: As shown in the video, stop the running task

E 7. 760: Create an AWS Fargate service. Watch the video: https://youtu.be/R1NW_M3jbjQ

- [Video timestamp: 0sec – 3min 10sec]: As shown in the video create an application load balancer which we can use in the service. Delete the default/temp target group it has created.
- [Video timestamp: 3min 10sec – 10min]: As shown in the video, create a service and run MyWolfApp website as a service. Test the website running behind the load balancer and make sure it's working fine.
- [Video timestamp: 10min – 12min 30sec]: As shown in the video, observe...
 - how the private IP address of the containers are mapped to the load balancer and check the health status of each container
 - how to get the logs of running containers
- [Video timestamp: 12min 30sec - 14min]: As shown in the video, stop a running task and simulate an application crash/disaster. Observe how the service automatically recovers after the crash.
- What's the advantage of running the website as a service as opposed to running it as a manually started task?



E 8. 770: Completing the final pipeline: Watch the video <https://youtu.be/O-iNLSilZZY>

Debugging tips: If your build pipeline gets stuck with the following error, restart the Linux build instance.
Error response from daemon: failed to start shim: exec: "docker-containerd-shim": executable file not found in \$PATH: unknown

- [Video timestamp: 0min - 5min]: As shown in the video, explore the pipeline and understand how the end-to-end configuration has been done
- [Video timestamp: 5min - 9min 30sec]: As shown in the video, modify the build tasks to tag images with correct repository URL and point the publication of the images to the correct repository. Attach the right permission to the build agent so that it can refresh the ECS service, and finally execute the pipeline.
- [Video timestamp: 9min 30sec-11min 50sec]: As shown in the video, observe how the container deployment happens behind the scene.
- [Video timestamp: 9min 30sec-18min]: As shown in the video, make some modification to the website code and commit it to the Azure repository and observe how the end-to-end integration works.

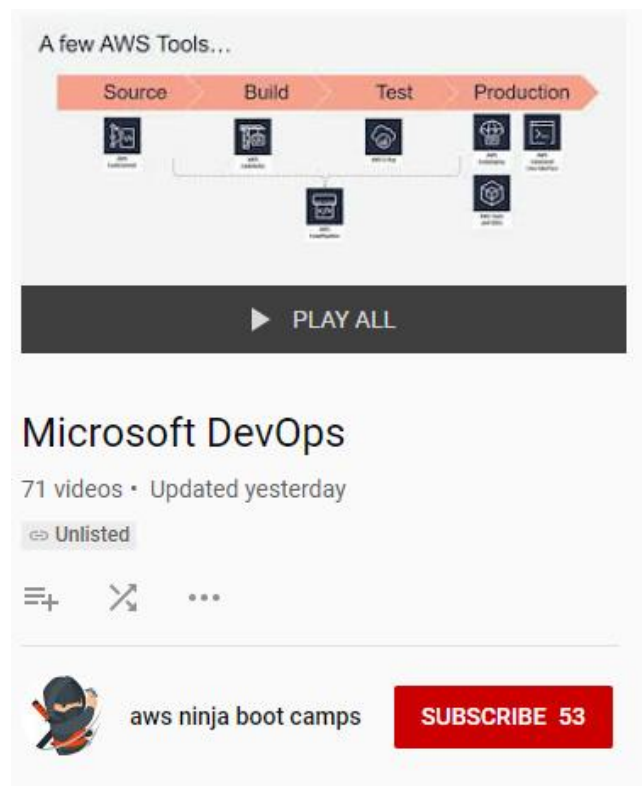
As shown in the video, complete the pipeline and get the pipeline to automatically deploy the containers into the cluster.

List of videos

All videos are available as a playlist at

<https://www.youtube.com/playlist?list=PLDvuRFHx48vBAJ4VjBb0lcqgYzv4yABXA>

Feel free to subscribe to the Ninja Bootcamp YouTube channel for deep dive hands-on technical content



You now have tools and knowledge to create
DevOps pipelines with Azure DevOps and integrate
them with AWS

Well done warrior!



Version 1.0

