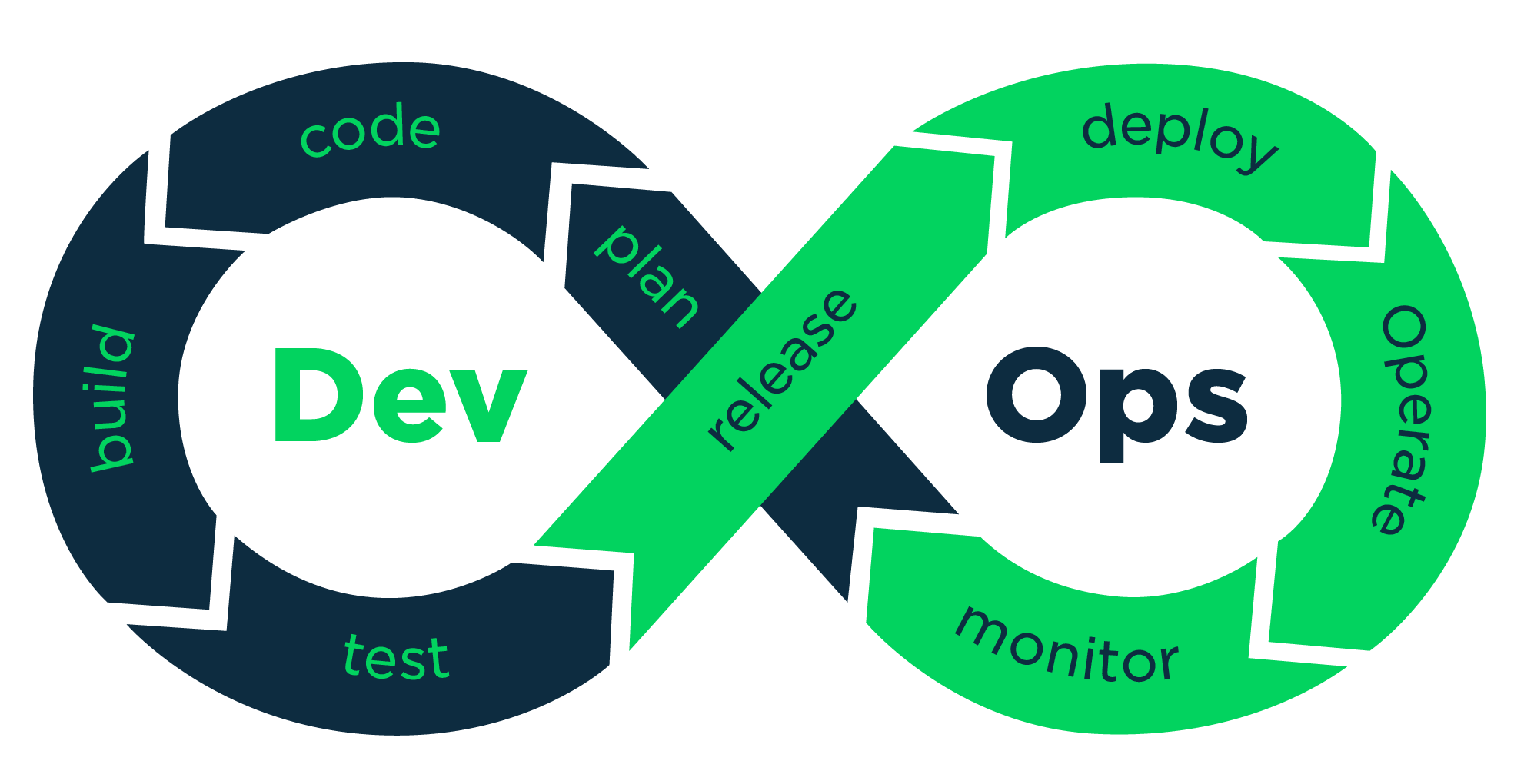
**Table of Contents:**

1. **Continuous Integration and Continuous Deployment**
2. **Tools and Services Used**
3. **Workflow Diagram**
4. **List of Services:**
5. **Provisioning Services**
6. **Security Services**
7. **Deployment Services**
8. **Monitoring Services**
9. **Decommissioning Services**



**Continuous Integration:** is a software development practice where continuous changes and updates in code base are integrated and verified by an automated build scripts using various tools.

**Continuous Deployment:** is an ongoing collaboration between the people who are part of the software creation process (Dev) and the people who are part of the release process (Ops).

**Scenario:**

We are planning to automate the entire Infra setup, Security, Deployment and Monitoring of Cora Analytics Application using Industry Standard DevOps Tools.

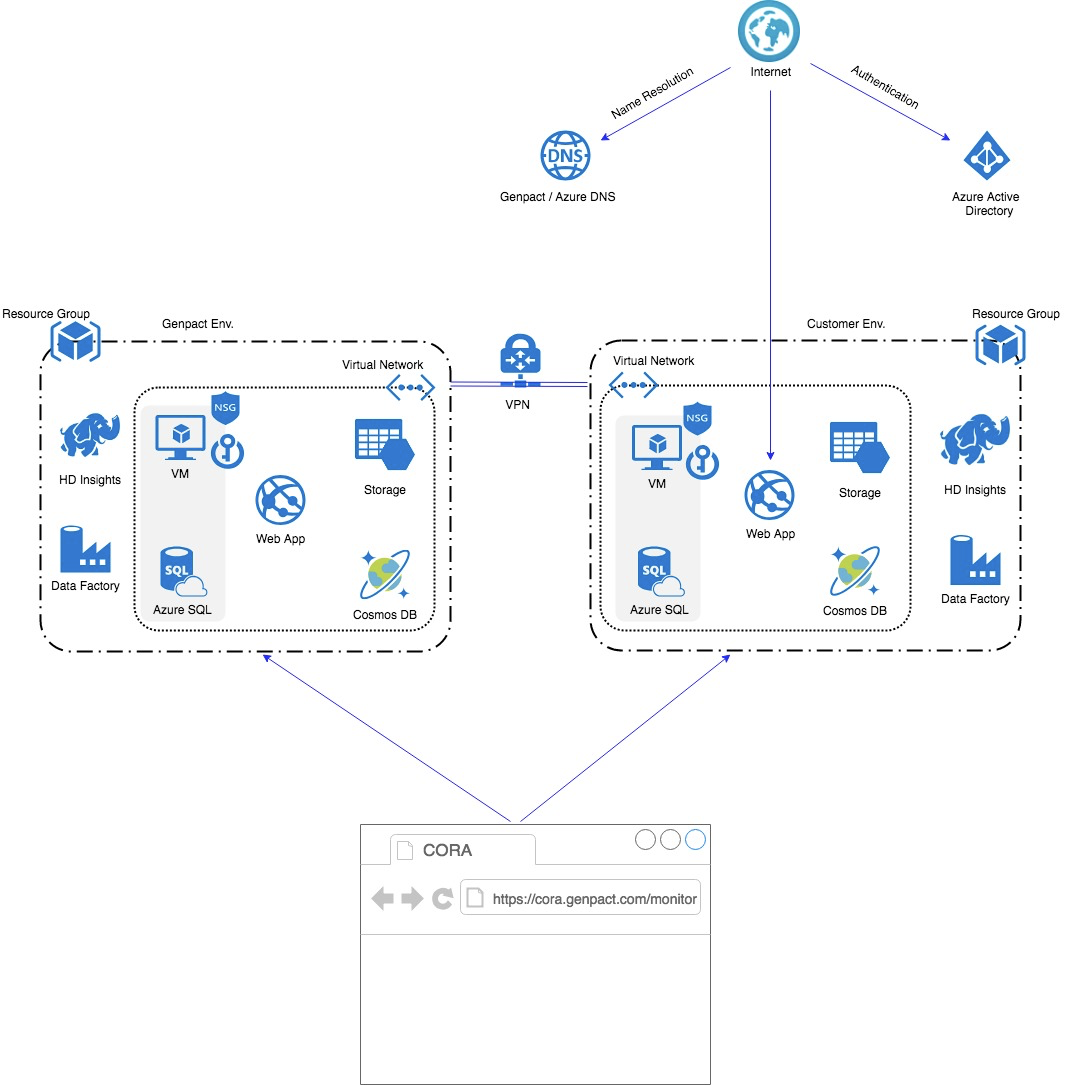
**Tools and Services used:**

1. Microsoft Azure
2. Jenkins
3. Ansible
4. Bash Scripting
5. Azure CLI
6. Azure ARM Templates and
7. Azure Runbook (Powershell)

|  |  |
| --- | --- |
| **Tools/Services** | **Purpose** |
| Microsoft Azure | It’s a **c**[**loud computing**](https://en.wikipedia.org/wiki/Cloud_computing) **service** created by [Microsoft](https://en.wikipedia.org/wiki/Microsoft) for building, testing, deploying, and managing applications and services through a global network of Microsoft-managed [data centers](https://en.wikipedia.org/wiki/Data_center). |
| Jenkins | Jenkins is an open-source **Continuous Integration** software tool written in the Java programming language for testing and reporting on isolated changes in a larger code base in real time. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds. |
| Ansible | Ansible is an open source automation platform. It is very, very simple to setup and yet powerful. Ansible can help you with **configuration management, application deployment, task automation**. It can also do IT orchestration, where you have to run tasks in sequence and create a chain of events which must happen on several different servers or devices. |
| Bash Script | It’s a [Unix shell](https://en.wikipedia.org/wiki/Unix_shell) and [command language](https://en.wikipedia.org/wiki/Command_language) written for the [GNU Project](https://en.wikipedia.org/wiki/GNU_Project) as a [free software](https://en.wikipedia.org/wiki/Free_software),  It’s a [**command processor**](https://en.wikipedia.org/wiki/Command-line_interpreter) that typically runs in a [text window](https://en.wikipedia.org/wiki/Terminal_emulator) where the user types commands that cause actions. Bash can also read and execute commands from a file, called a shell script. |
| Azure CLI | It’s your **command line for Microsoft Azure** that you can use from anywhere. With CLI, you can create, manage, and delete services on the command line via cmd.exe, bash or {your shell} on the operating system of your choice. |
| Azure ARM Template | An Azure Resource Template is a JSON file. These templates can be used to **deploy resources**. You can deploy the template over and over again without affecting the current state of resources |
| Azure Runbook/Azure Automation | This feature gives administrators the ability to create an Azure runbook to manage cloud resources more easily. Azure Automation is a native Microsoft tool to run **Windows PowerShell** platform as a service (PaaS) |

**Workflow Diagram:**

**Note:** The architecture given below is just a view of application from DevOps perspective and not the exact architecture of the Application.

****

**Tasks:**

1. **Provisioning Services**:

**i. Work Done:**

Provision the infrastructure including all the necessary services (PAAS, IAAS) required for the Application to run. Currently, there are 3 environments: QA, Non-Prod, PROD

We are at the beginning phase and we are Concentrating on Staging Env. Please find the list of resources provisioned below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Resource Name** | **Tools used** | **Service** **Models** | **Status** |
| Virtual Machine | Azure CLI | IaaS | Done |
| SQL Server | Azure CLI | PaaS | Done |
| Storage Account | Azure CLI | PaaS | Done |
| Azure Web App | Azure CLI | PaaS | Done |
| Virtual Network | ARM Template | PaaS | Done |
| HD Insights | Runbook (powershell) | PaaS | Done |
| DataFactory | ARM Template | PaaS | Done |
| Azure Key vault | Azure CLI | PaaS | Done |
| Cosmos (Gremlin DB) | Runbook (powershell) | PaaS | Done |
| Cosmos (JSON Store) | Azure CLI | PaaS | Done |
| Azure Active Directory | Azure CLI | PaaS | Pending |

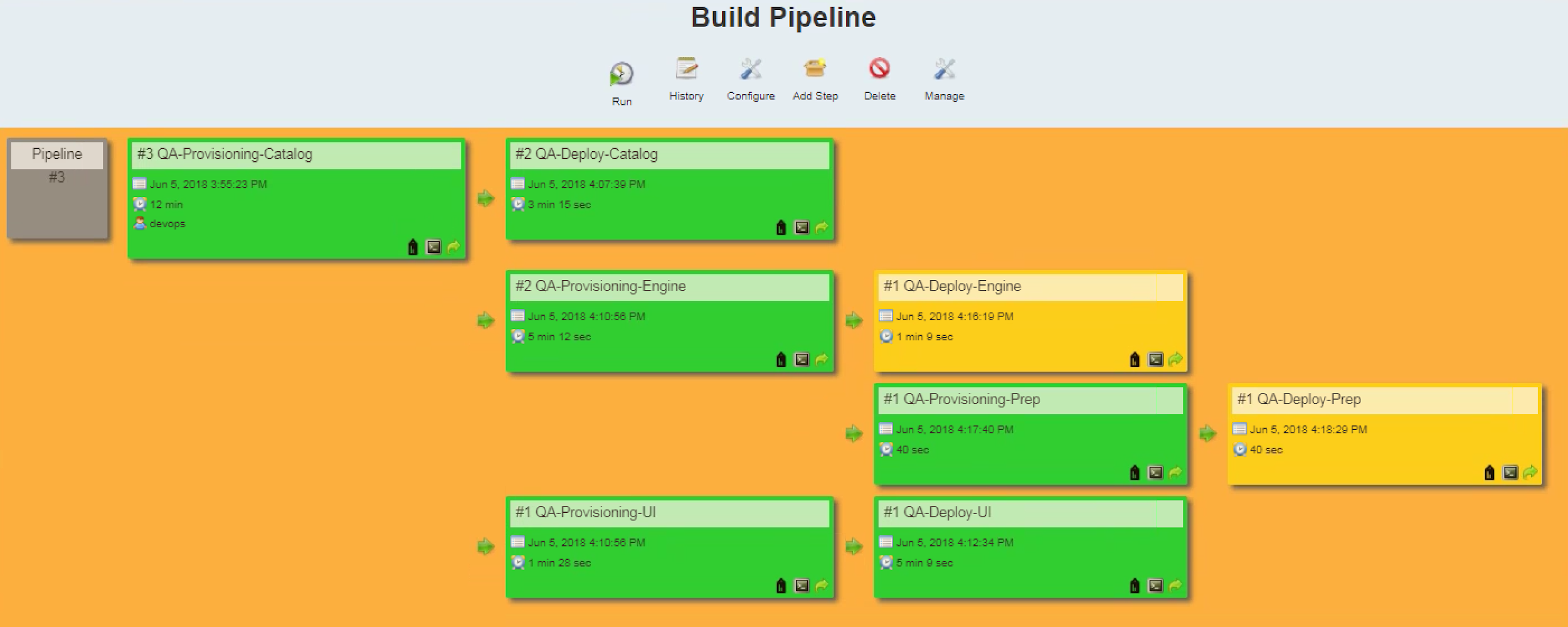
**Jenkins job for Provisioning infra with Parameters:**

Find the list of parameters below used for provisioning the environment.

|  |  |  |
| --- | --- | --- |
| **Application** | **Parameters** | **Comments** |
| Catalog | VMSize | Virtual Machine configuration |
| AppSize | Azure Web App configuration |
| DBEdition | SQL DB Size and configuration |
| DTU |
| vCores |
| SQLEncryption | SQL DB Encryption using TDE - (Yes/No) |
| DiskEncryption | Virtual Machine Storage Encryption - (Yes/No) |
| Throughput | Gremlin DB configuration |
| HDInsights | Cluster configuration |
| Username | Login Credentials for HDInsights, SQL DB, VM.  (Script written in a way to fail, if Password doesn’t match) |
| Password |
| ConfirmPassword |
| SoftDelete | Save and Recover data when blobs are deleted - (Yes/No) |
| Data Prep | AppSize | Azure Web App configuration |
| SoftDelete | Save and Recover data when blobs are deleted - (Yes/No) |
| Core Engine | AppSize | Azure Web App configuration |
| UI | AppSize | Azure Web App configuration |

The configured jenkins Pipeline will provision the resources and deploy the source code for all the four Applications.

* Catalog
* Data Preparation
* Core Engine
* Cora UI



**Steps Involved:**

The above diagram illustrates the fresh provisioning and deployment of CORA Application.

**JOB 1 - Catalog App:**

1. Once the user passes the parameters through jenkins, the pipeline is initiated.
2. The First Job will pull the code from Git Repo, then provision the resources and make changes to a reference file with newly created resources.

For Example,

We are provisioning SQL DB, The endpoint and credentials of SQL DB should go inside the application, so for that Developer has created a reference file within the source code which will be like “Key:Value” pairs. DevOps Team will just change the Value field in the file and the application will refer to the Key and take the updated new value into the application while building.

**Original value** in the file (graphdb\_init.properties):

sql.jdbconnectionstring=jdbc:sqlserver://xyz.database.windows.net:1433;database=zyx;user=devops@supplychainanalytics;password=DevOPS@435;encrypt=true;trustServerCertificate=false;hostNameInCertificate=\*.database.windows.net;loginTimeout=30;  
  
**Updated value** in the file (graphdb\_init.properties):

First, we find the sql db name and sql server name using the CLI and update the same in the file using sed command.

**Parameters:**

* **servername**=$( az sql server show -n gecastgsqlsrv$label -g PhoenixCatalogStg --query name | sed 's/\"//g')
* **fqdn**=$( az sql server show -n gecastgsqlsrv$label -g PhoenixCatalogStg --query “fullyQualifiedDomainName” | sed 's/\"//g')
* **dbname**=$(az sql db show -n gecastgsqldb$label -g PhoenixCatalogStg -s gecastgsqlsrv$label --query name | sed 's/\"//g')
* **${13}** = Username - will be given as 13th parameter in jenkins, You can refer the above table for parameters details.
* **${14}** = Password - will be given as a 14th parameter in jenkins

sed -i "3s/sql.jdbconnectionstring.\*$/sql.jdbconnectionstring=jdbc:sqlserver:\/\/$dbserver\:1433\;\database=$**dbname**\;user=$**{13}**@$**servername**\;\password=$**{14}**\;\encrypt=true\;trustServerCertificate=false\;hostNameInCertificate=\*.database.windows.net\;loginTimeout=30/g" CAP-Data-Governance-API/src/main/resources/graphdb\_init.properties

**JOB 2 - Catalog App:** Once job 1, completes the provisioning, job 2 will be triggered subsequently and it will

* Build the code for Catalog Application with update reference file
* Upload code to archiva ,
* Upload code to SonarQube and
* Deploy to the Web App.

**JOB 3 - Core Engine App:**

Once job 2 done, again provisioning for the application Core Engine will be started and it will do the same as job 1 like provisioning and updating the reference file.

**JOB 4 - Core Engine App:**

Job 4 will do the same as job 2, but it’s for Engine App.

* Build the code for Catalog Application with update reference file
* Upload code to archiva ,
* Upload code to SonarQube and
* Deploy to the Web App.

**JOB 5 - DataPrep App:**

Wait till job 4 completed, again provisioning for the application Data Prep will be started and it will do the same as job 1 like provisioning and updating the reference file.

**JOB 6 - DataPrep App:**

Job 6 will do the same as job 2, but it’s for DataPrep App.

* Build the code for Catalog Application with update reference file
* Upload code to archiva ,
* Upload code to SonarQube and
* Deploy to the Web App.

**JOB 7 - UI App:**

Later, Once job 6 completed, again provisioning for the UI application will be triggered and it will do the same as job 1 like provisioning and updating the reference file.

**JOB 8 - UI App:**

Job 8 will do the same as job 2, but it’s for UI App.

* Build the code for Catalog Application with update reference file
* Upload code to archiva ,
* Upload code to SonarQube and
* Deploy to the Web App.

**ii. Work in Progress / to be done:**

Leveraging the same script and integrating with other environments like QA, Dev, etc.. There will be few modifications as per the environment.

**iii. Installation verification:**

Once the resources are provisioned, We are going to verify the Provisioned/Deployed resource to make sure they are in place. Most of the resources are verified by checking the status using azure cli.

Let’s say, For Cosmos DB:

We are using a simple while loop with cli command to check the status and if not success, check for every 10 seconds till it exhausts another 10 mins of time.

**# /bin/bash**

**datetime=$(date +"%Y%m%d\_%H:%M")**

**msg() {**

**local message="$1"**

**echo "[$datetime] [INFO] $message"**

**}**

**error\_exit(){**

**local message="$1"**

**echo "[$datetime] [ERROR] $message"**

**exit 1**

**}**

**secs=600  
seconds=0  
while (( seconds < secs ));  
do  
 status=$(az cosmosdb show --name gecastggr$label --resource-group PhoenixCatalogStg --query provisioningState | sed 's/\"//g')  
 expected=Succeeded  
 if [ "$status" = "$expected" ];then  
 msg "Gremlin DB provisioned"  
 break;  
 else  
 sleep 10  
 if [ $secs = $seconds ]; then  
 error\_exit "Gremlin DB provisioning failed"  
 fi  
 fi  
done**

**#Output (Either one of the below):**

**[20180625\_06:37] [INFO] Gremlin DB provisioned**

**(or)**

**[20180625\_06:14] [ERROR] Gremlin DB provisioning failed**

**Running Docker Containers - POC:**

We have also tried setup of docker containers as a replacement for Web Apps and pointed it to Load balancer.

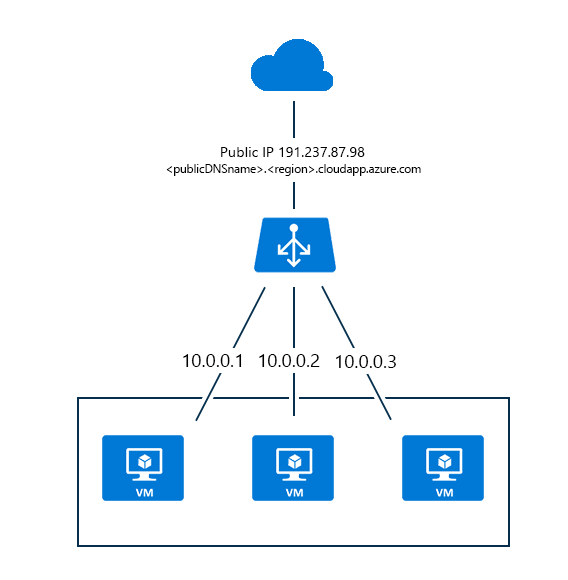
Below are the steps involved.

* Ubuntu docker image is used as a base image here.
* Configured tomcat service in the container and built an image out of it. The container will start running the tomcat service on the specified port (ex. 8080).
* We will build the Code in host machine and make it as a war file using gradle.
* We have written the Dockerfile, where it will check for the built code (.war) and copy it to docker container (where Tomcat is running) and build a image out of it.
* Azure VM will be deployed with updated image and latest application code, which is placed on the private network.
* We have configured a Load Balancer (public network) in front of Azure VMs to forward/proxy the traffic to tomcat, which will serve the web page for end user.

The load balancer is created to route traffic among multiple virtual machines. A public IP address is an individual resource that has a domain label (DNS name) and the public IP address is associated with the load balancer resource.

Load balancer rules and inbound NAT rules use the public IP address as the Internet endpoint for the resources that are receiving load-balanced network traffic.

Docker containers (tomcat) will be hosted on the every single Azure VM.



**b.** **Security Services**:

**i. Work Done:**

Once the infra is provisioned the next thing is to secure the environment. It needs to be done at both Service and Host level.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resource Name** | **Firewall** | **Encryption** | **Patching** | **Status** |
| Azure Key Vault | - | - | - | Done |
| Virtual Machine | **Pending -** Only authorized networks allowed through Security Group. | **Done** at Volumes(Disk) level using Custom Keys. | Will be taken care by Genpact’s AWS Team. | Partially Done |
| SQL Server | **Pending** - Only authorized networks allowed through Security Group. | **Done** - TDE Encryption using Custom Keys. | It’s a PaaS Service. | Done |
| Storage Account | **Pending -** Only authorized networks allowed through Security Group. | End-to-End Encryption by default (Microsoft) | - | Done |
| Virtual Network | Resources will be provisioned on Private Subnet (No public internet access) | - | - | Done |
| Cosmos DB | **Pending -** Only authorized networks allowed through Security Group. | End-to-End Encryption by default (Microsoft) | - | Partially Done |
| Azure Web App | **Pending -** Only authorized networks will be allowed. | SSL enabled by default | - | Done |
| Azure Active Directory | **-** | - | - | Pending |

**ii. Work to be done:**

Integration of Azure Active Directory for Authentication.

**c.** **Deployment Services**:

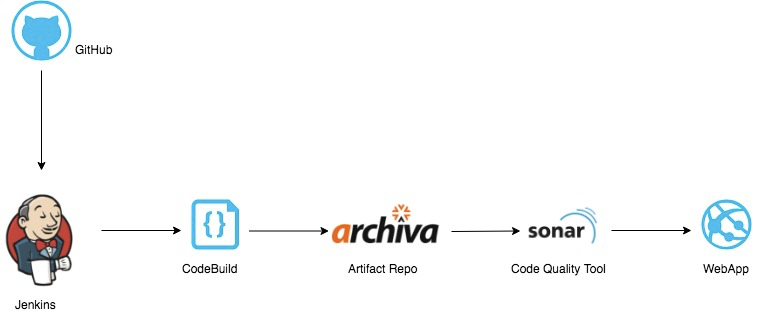
**i. Work Done:**

Currently the deployment is done manually using FTP and it’s a time taking process.

We have automated the deployment using kudu api request through curl command with authentication and secure communication (https).

The CI/CD for the deployment is done through jenkins.

**ii. Workflow Diagram:**



**iii. Workflow:**

1. Developer will push/commit the code to Git Repo.
2. Jenkins will keep polling the Repo for every minute using the Poll SCM feature, We need to provide the username,password or SSH keys for authentication.
3. Once jenkins detects there is a change in source code, the job will be triggered immediately.
4. Code will be pulled from the repo to jenkins workspace (just a directory) using the authentication method provided.
5. Jenkins will execute the script and actions provided by us in the job one by one.
6. The Script will build the code downloaded appropriately based on the building mechanism.

Let’s say for Java, It’s :

gradle publish

for nodejs. It’s

npm install

npm run build:ssr

1. Then, the code will be uploaded to Apache Archiva (Artifacts repository) for backup and future reference.
2. Also the code will be sent/uploaded to the SonarQube Tool for Code Quality Analysis using the Sonar plugin in jenkins.
3. Ensuring the code uploaded to sonar, the Code will be deployed to Azure Web Application using “POST” method provided by Azure kudu project API’s which requires authentication and establishes a secure handshake.
4. Post deployment, The Application will be restarted and the application will be accessible through “HTTPS”.

**iv. Work to be done:**

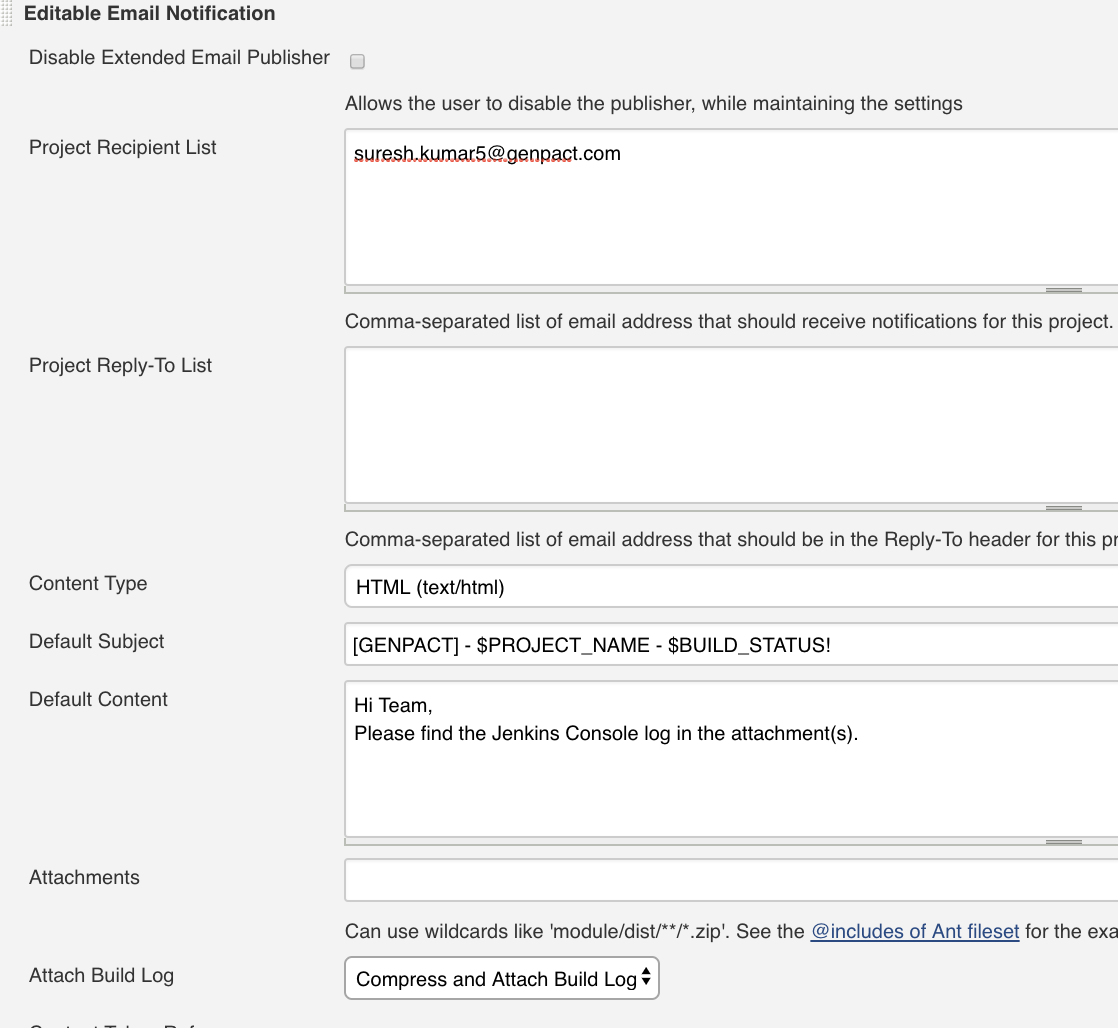
Implement the same solution for all environments once the workflow is approved

**d.** **Monitoring and Notification Services**:

**i. Work done:**

1. **Email Notification:**

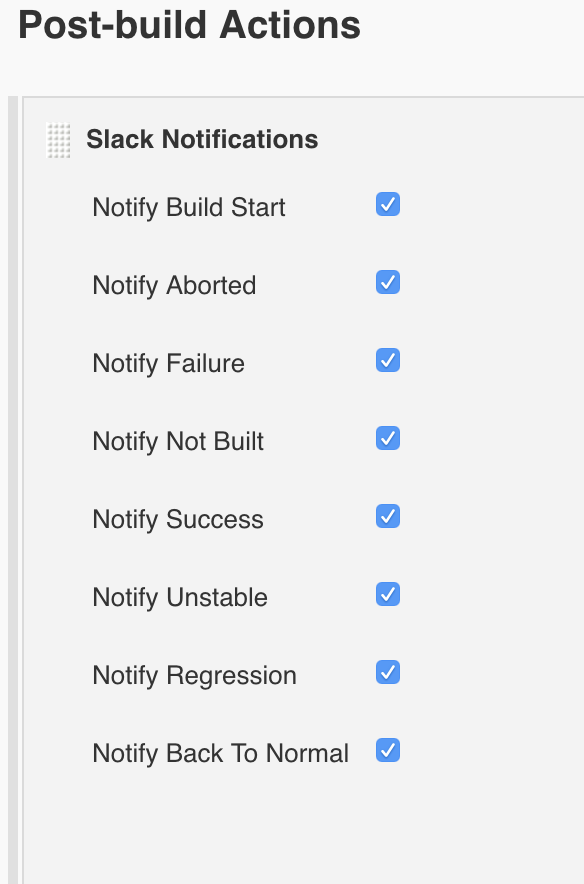
We are sending email notification to users through jenkins email plugin.



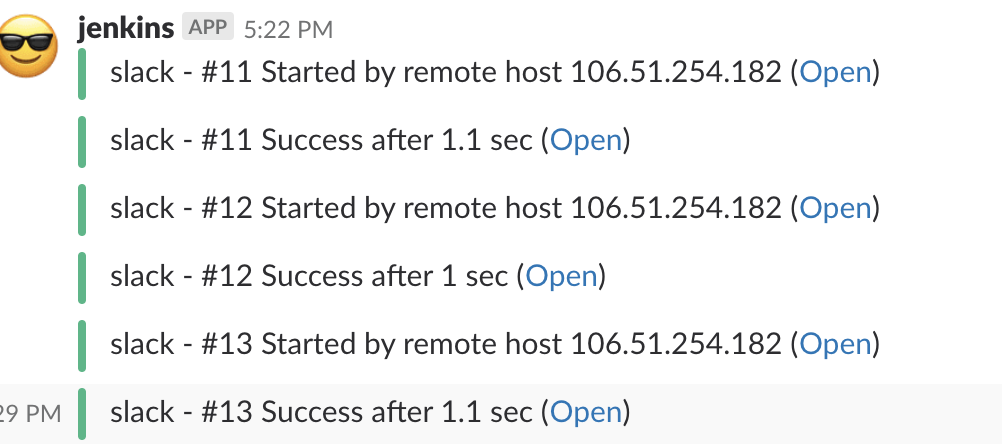
1. **Slack Notification:**

Configure the slack notification plugin and Global Slack Notifier Settings with slack token for authentication.

Configure the jenkins job with slack plugin and enable the notification. Then, after every successful or failed build , the notification will be sent to slack channel.



You can click on open in the message received and it will redirect you to jenkins console (if accessible from public internet).



**i. Work to be done:**

1. Install/Integrate third party Monitoring Tool and Azure Monitor Service with all IaaS/PaaS Resources using automation tools like ansible and Scripting.
2. Install Trend Micro agent in all VM’s using ansible.
3. Trend Micro will take care of the vulnerability and keeps scanning the environment on regular basis.

**e. Decommissioning Services:**

**i. Work to be done:**

Some of the environments should be decommissioned after it’s been utilized, For Ex. Dev env. it’s not required to be always ON.

By Design, We are provisioning the environment (let’s say: Dev) on Resource Group (Logical grouping) basis in Azure. We can terminate the entire environment by just one-click (Delete Resource Group) to discontinue the environment.

That is a straightforward method, Decommissioning of specific resource or service as per the architecture in future might require some customization.