bTranslator

Brainy Translator System

Technical Design Document

Version 1.0

Abstract

Low Level Technical Solution & design details enlightening on the application architecture, framework components, technologies involved, application infrastructure and other related aspects

Author

Rishi Raj Bansal

ThoughtExecution

Solutions with Commitment, Quality & Trust

Document Control

Version History

Date	Version	Amendments	Updated By
14 January 2020	1.0	Initial Draft	Rishi Raj Bansal

Table of Contents

1. INTRODUCTION

1.1 Overview

A text translation application that uses advanced machine learning technologies to provide high-quality translation to translate raw text or unstructured text documents in multiple languages.

It provides translation between a source language (the input language) and a target language (the output language). It uses semantic representation to generate a translation one word at a time.

It's a great solution in cases where the volume of text is high, speed is critical, and very minor level of translation imperfection is acceptable.

1.2 Purpose of Document

This document provides the low level technical design of the Translate system that is developed for automated language translations. It elaborates on the implementation details, the architecture of the application and all the components that are developed as part of implementation. The purpose of this document is to describe in sufficient detail how the system is constructed. It identifies the top-level system architecture, and identifies hardware, software, communication, and interface components.

1.3 Acronyms & Glossary

1.3.1 Acronyms

#	Acronym	Description
1.	MVC	Model View Controller
2.	SOA	Service Oriented Architecture
3.	API	Application Programming Interface
4.	EC2	Elastic Compute Container
5.	EBS	Elastic Block Storage
6.	ELB	Elastic Load Balancing
7.		
8.		
9.		

1.3.2 Glossary

#	Glossary	Description
1.		
2.		
3.		
4.		

2. ARCHITECTURAL DESIGN

2.1 Overview

This section elucidates the overall architectural design of the system with the brief description of the progression flow among the systems and the interfaces involved in the program.

The architecture of the system is based on **MVC** and **SOA** design principles.

MVC is a software architecture that separates domain/application/business logic from the rest of the user interface. It does this by separating the application into three parts: the model, the view, and the controller.
The *model* manages fundamental behaviours and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, and even to notify observers in event-driven systems when information changes. This could be a database, or any number of data structures or storage systems.
The *view* effectively provides the user interface element of the application. It renders data from the model into a form that is suitable for the user interface.
The *controller* receives user input and makes calls to model objects and the view to perform appropriate actions.

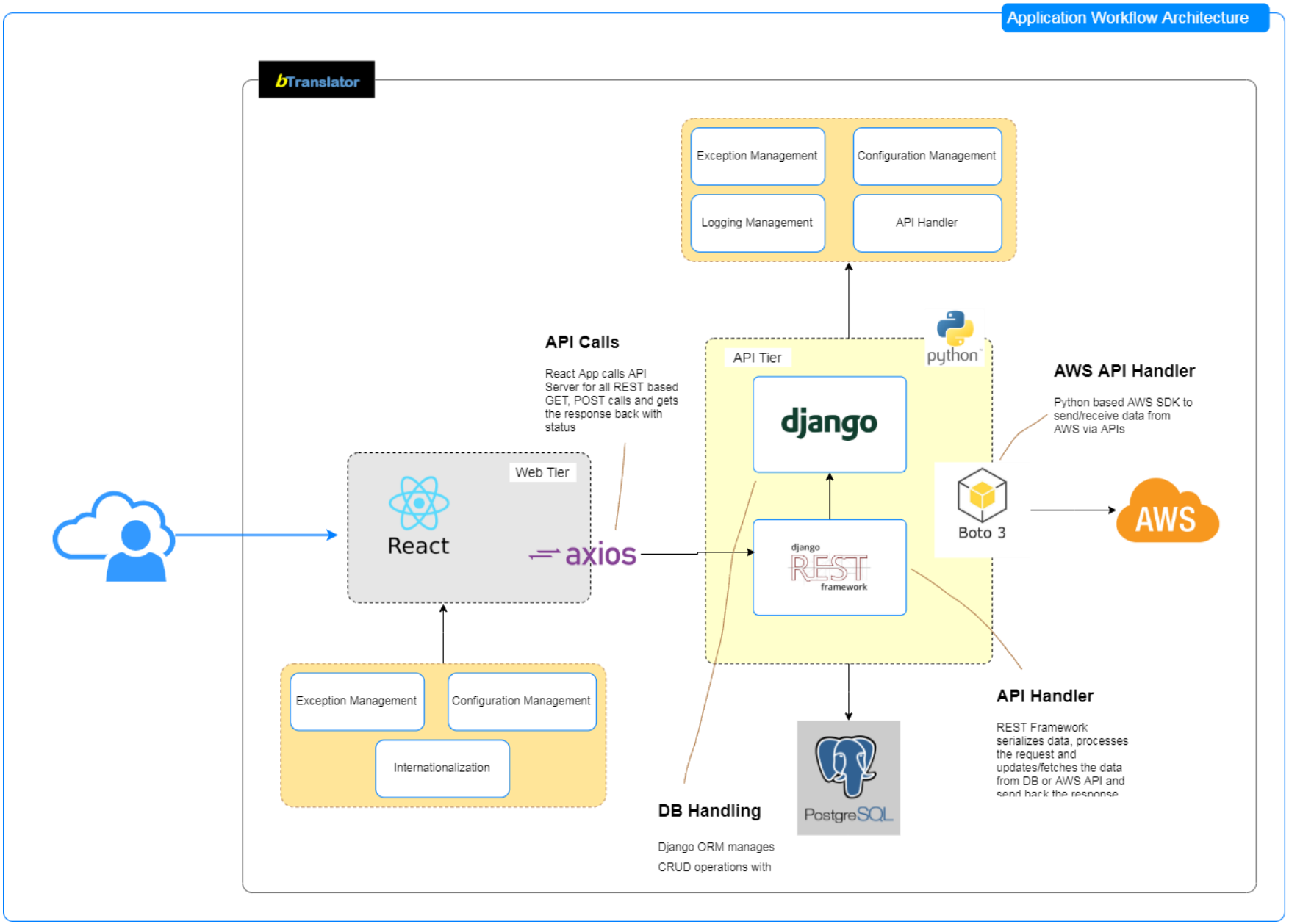
SOA (Service-oriented architecture) is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.
It's an approach used to create an architecture based upon the use of services. Services (such as RESTful Web services) carry out some small function, such as producing data, validating a customer, or providing simple analytical services.
In addition to building and exposing services, SOA has the ability to leverage these services over and over again within applications (known as composite applications). SOA binds these services to orchestration, or individually leverages these services. Thus, SOA is really about fixing existing architectures by addressing most of the major systems as services, and abstracting those services into a single domain where they are formed into solutions.

One of the keys to SOA architecture is that interactions occur with loosely coupled services that operate independently. SOA architecture allows for service reuse, making it unnecessary to start from scratch when upgrades and other modifications are needed. This is a benefit to businesses that seek ways to save time and money

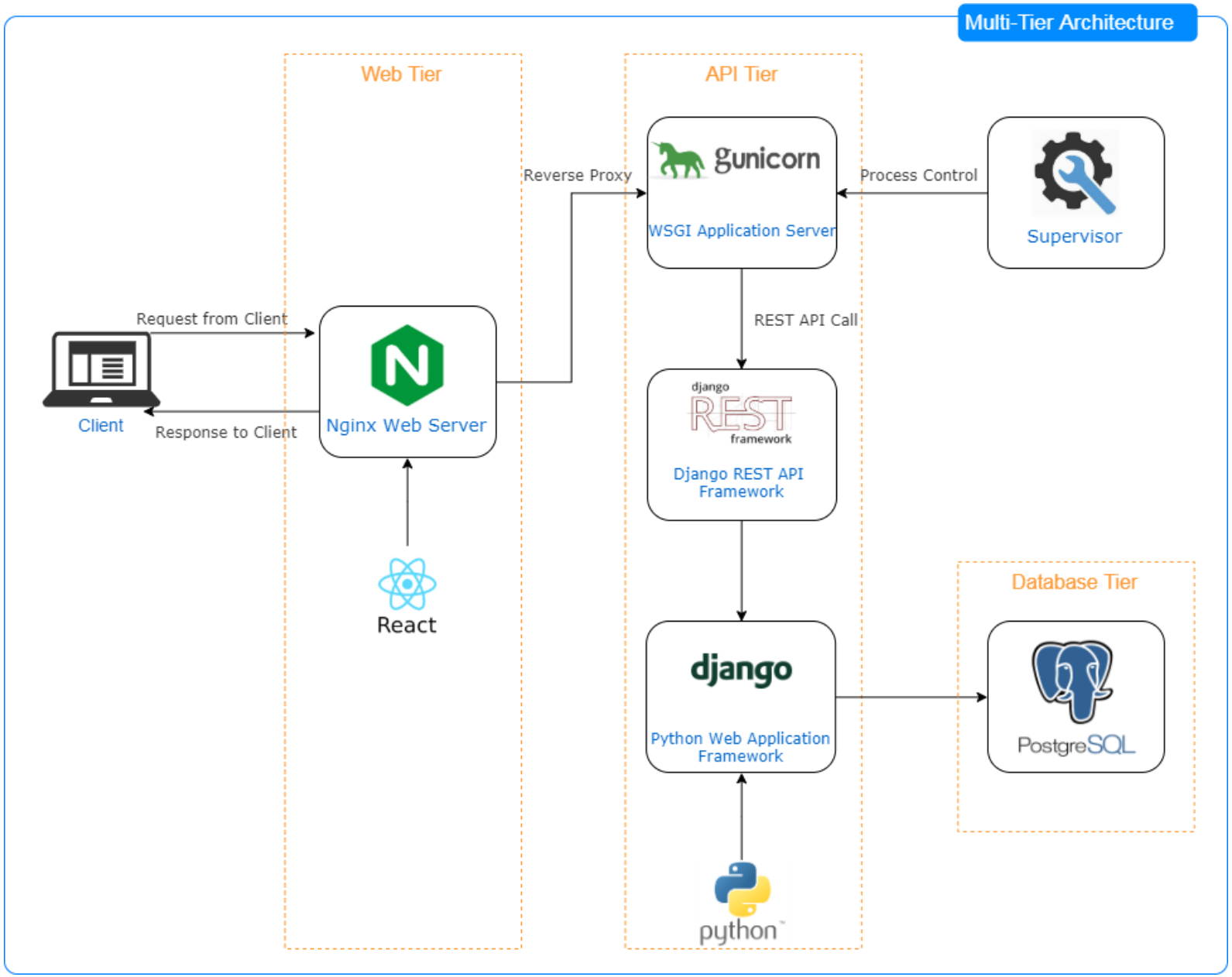
The application workflow architectural diagram of the program has been depicted in **Workflow Architecture** Diagram which illustrates all the systems and the interfaces that are involved in the system and defines the process and data flow between them in logical mode.

The architecture of the system is based on n-tier and the request from the client goes through one layer to another layers before the process of request is completed. **Multi-Tier Architecture** depicts the components in the corresponding layers where they resides.

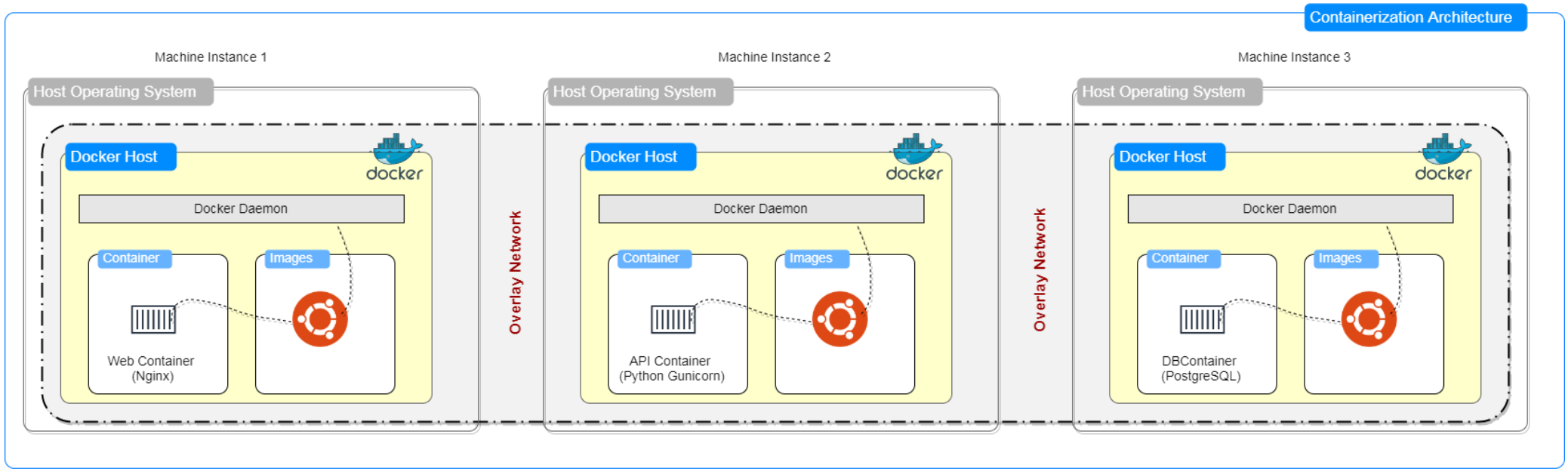
2.2 Application Workflow Architecture



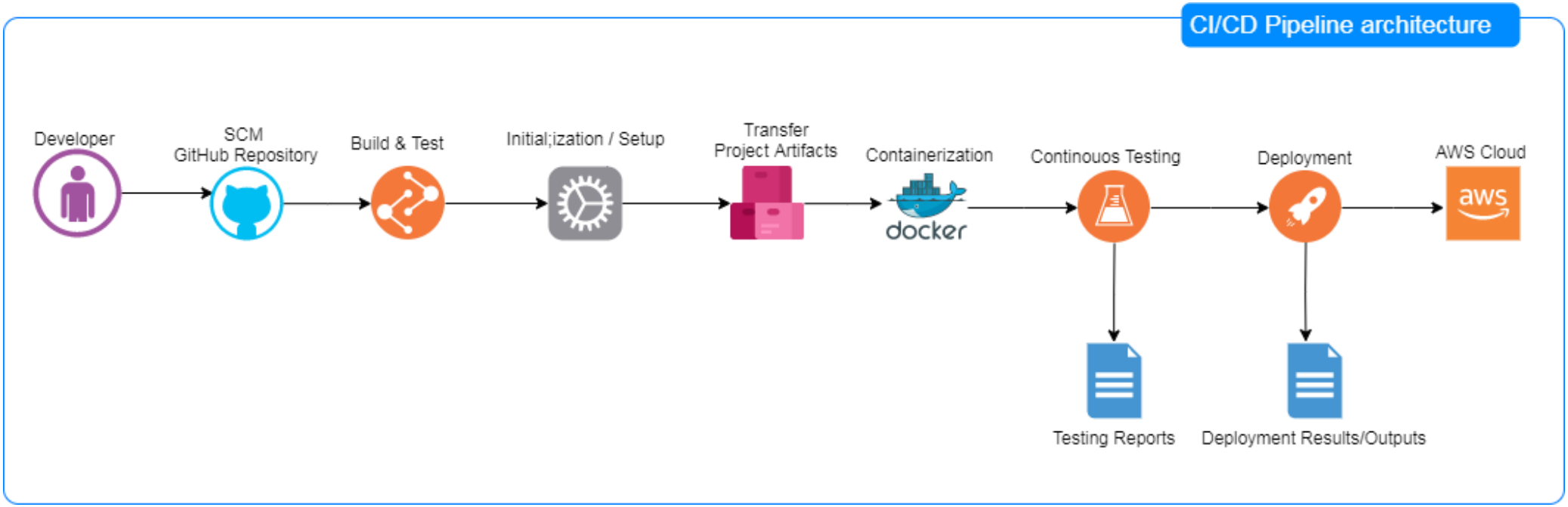
2.3 Multi-Tier Architecture (Including Servers)



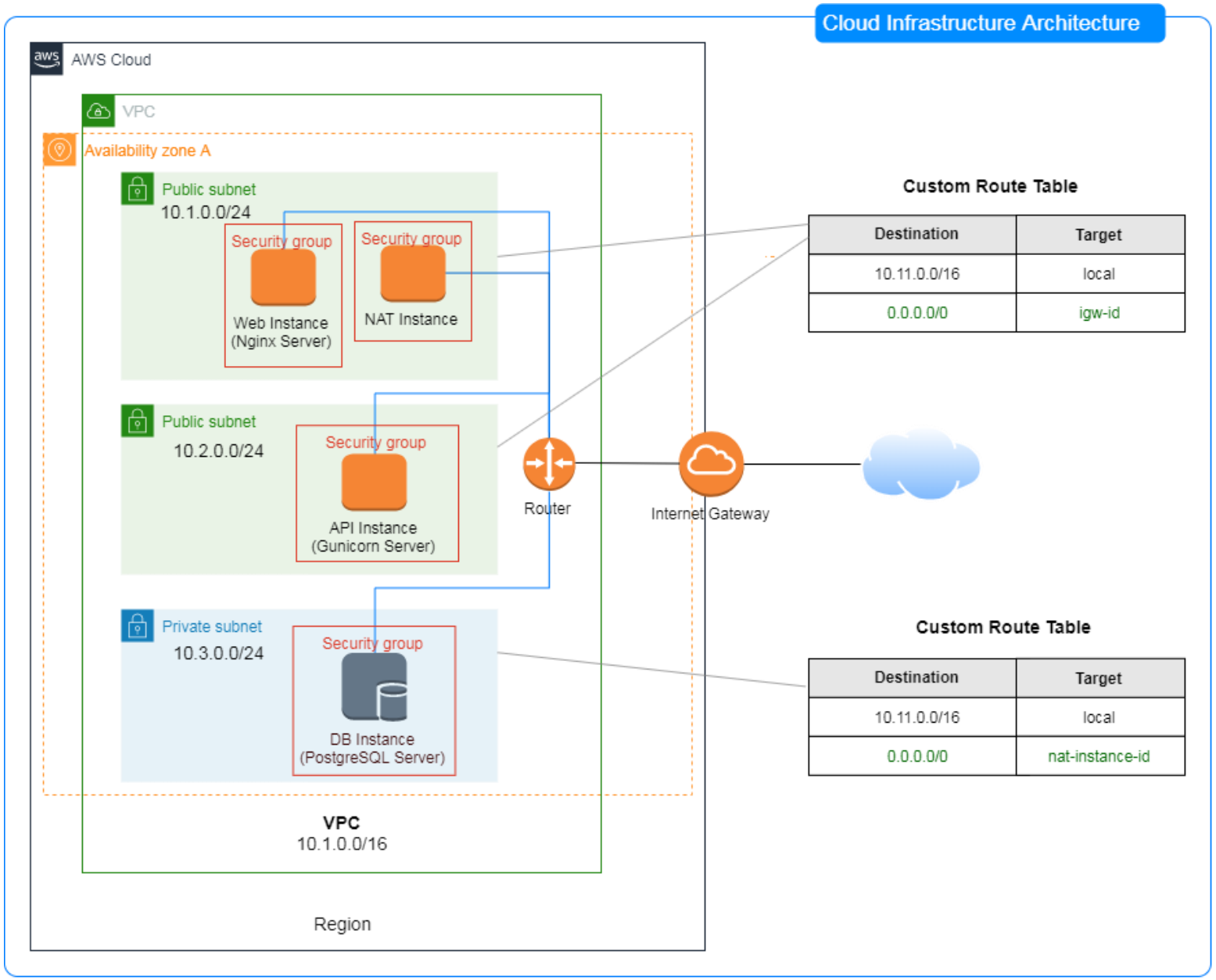
2.4 Containerization Architecture



2.5 CI/CD (Jenkins) Pipeline Architecture

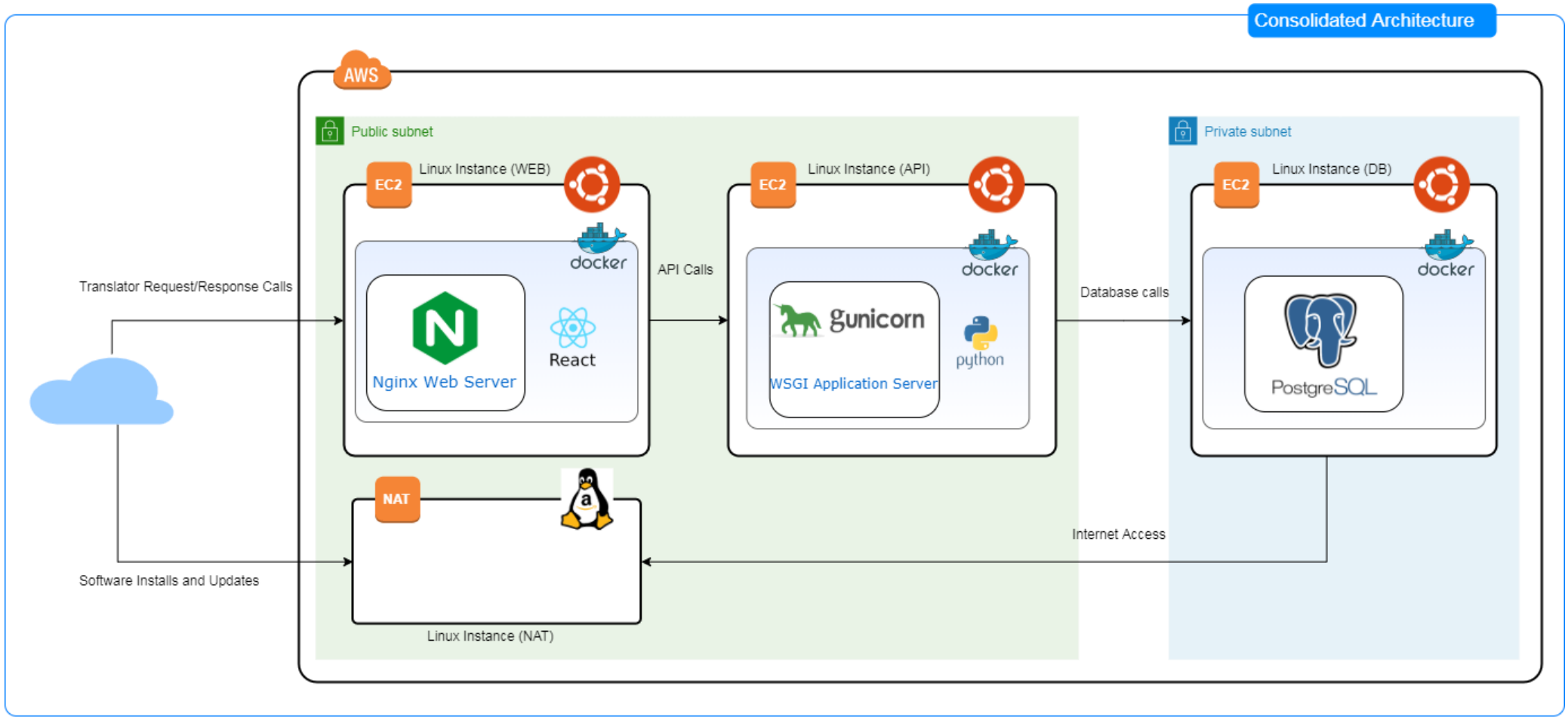


2.6 Cloud Infrastructure Architecture



2.7 Consolidated Architecture

Following architecture depicts the overall synopsis of all the infrastructure components involved in the application deployment.



2.8 Functional Components

This application is structured with multiple functional components that works collaboratively to make application functioning successful.

Few of the major components implemented in the application are defined below:

- **Dependencies Management**
  For Frontend Tier
  All required dependencies are managed within the application level inside node_modules component. All these dependencies are packaged in the file package.json with the respective versions. This makes very convenient for application deployment and migration. On application packaging, all these dependencies become part of the application and thus, no need of external installations during the application deployment.

  For Backend (API) Tier
  All libraries dependencies of Python application is freeze in requirements.txt file which is later used to install all required libraries on the system where the application will be deployed. No explicit installation is necessary to install any dependency.

- **Configuration Management** – This is one of the important component of the application which manages overall configuration of the application.
  - This component initializes only one time on the application start-up and loads all the configurable properties for all modules and components in memory objects.
  - Each configuration property is classified into the module to which it belongs and are stored in memory object in likewise format. All config properties are cached in memory objects.
  - Fetched configuration data is not get stored in a single memory object, eventually it is stored in multiple collection objects based on the classification of configuration data. By classifying the data, it enhances the performance of the application by making the searching of the desired data faster in the memory objects as compare to find in the single memory object.
  - This increases the overall performance of the application by saving numerous file hits.

  If it needs to update / modify the value of any configurable data, then it would only require to update the relevant configuration key in the config file and no code level changes.

- **Logging Management** – Customizable Logging mechanism implemented in the framework for logging the application activities for *info* or *debugging* purposes. These logs can be displayed on console or can be stored in text files.
  - All properties of Logging component is configurable and can be changed anytime by just updating the relevant key/value pair in the config file, no code level change is needed. Every property of Logging is configurable.
  - Each log display complete information with the timestamp, module name, file info, debug level etc.
  - For text files logging, rotating file mechanism is implemented to rotate the log into new file once the allowed threshold of the file size is accomplished. No. of files and file size is configurable.

- **Exception Management** – This is a most important and significant component for any application, so maximum emphasis is given in implementing completely shielded application by robust exception handling.
  - Custom exceptions classes are created for each type of business failure scenario to clearly identify what issue occurred during the request processing. All exceptions are inherited from

node.js superclass 'Error'. Primarily, these exceptions are classified at : Generic, Business, Controller and Global Error Handler levels.

- All exceptions are classified at framework's components level, that means, each component has their own subset of exceptions inherited from one main 'ApplicationException' class.
- In case of any exception, Exception Detail object is created which consists of all possible details like: Status, Code, Error Message, User Message and Stack trace. All these details are appeared in structured format on user interface, if any exception occurs at application level.
- Exceptions are also handled for all asynchronous level processing, Promises, callbacks handlers etc.

⇨ **Data Access Management** – This component is responsible for interacting with the database. All business requests related to CRUD operations are passed from this component. This also acts as a 'Model' layer of MVC. All database operations are handled at DAO (Data Access Objects) layer and results is sent back to business layer. Each Data object or Model represents a table (or collection) in the database. It creates connection to the application's database, processes the database requests and closed down the database handles after request processing.
All database connections properties are configurable and are kept in single config file.

⇨ **Business Validations** – This component is responsible for validating request data before sending it for processing. For each request there may be certain prerequisites or data that are essential for successful request processing and in absence of this data the request may fail, so to avoid these kind of scenarios business validations checks are applied to verify if all required data is present in the request's payload. These business validations checks are applied at 2 levels:

- *Client-end* – Custom validations checks at UI level are implemented on each UI control that are applicable for validation. On form submit, these validation checks are applied on each UI control and if it get fails an error message appears for each control for which the validation failed. Without passing all validations the form will not be allowed to submit.
- *Server-end (API)* – Even if the client end validations are surpassed, server end validations are applied on the request data. So, it provides dual security for business validations. These validations are handled by Django Rest Framework using Object Serialization which inspect each REST service call request's attribute against its corresponding value objects' property

⇨ **Internationalization** – This application supports localization and internationalization. That means, the same framework based application can be used for multiple languages and locals for different geographies.

- All locale based details can be configured into the config file and then application itself will look out all the details during the request processing.
- All strings (messages appeared on UI) are configurable in messages config file. Each Message config file represents one language and there can be as much messages config file as needed as per language. While rendering the view on UI, application will look into the required message config file to extract the messages based on the requested language type.

⇨ **Static Files Management** – To prevent the server showing location of static files and preventing the malicious attacks for unauthorized file requests attacks, middleware is implemented which manages all the static files from single location and provides all required security to prevent unauthorized file request attacks. All static files are redirected from web server and uses caching mechanism to increaser the performance.

⇨ **Security** – The framework is designed and equipped with all standard security standards, all requests are filtered by certain security levels. Few of them are :

- *Session validations* – In order to prevent unauthorized access to the application and secured forms of the application, session validation is implemented.
- *Cross Site Request Forgery* – is used in the application to prevent CSRF.
- *CORS* - This allows in-browser requests to the application from other origins with configured allowed locations.
- *SQL Injection* – DAO layer of the application implicitly manages unanticipated SQL injection attacks.

⇨ **REST API Framework –** Django Rest Framework (DRF) is used for REST APIs implementation.

- Serializers of DRF is used to have complete control on request. Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into JSON, XML or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data. Serializers are able to validate the incoming request parameters and also can have method on individual parameters to have specific checks.
- Class Based Views is used which defines get and post method. A paramter will be passed in request to identify what kind of services is being called and then in post method based on the service type corresponding action will be executed.

⇨ **Handling settings for different environments –** To keep application settings or configurations isolate from each different environment (dev, QA, Prod), .env files are used which store environment specific settings. Each environment has its own .env file with corresponding configuration settings. Application reads the key-value pair from .env file and adds them to environment variable. It is great for managing app settings during development and in production using 12-factor principles.

⇨ **Asynchronous calling of REST APIs –** Frontend tier calls the API tier using REST API calls to retrieve the desired response. These calls are asynchronous calls which are implemented using 'Axios' library. Axios is promise-based library and thus we can take advantage of async and await for more readable asynchronous code. It can also intercept and cancel requests, and there's built-in client side protection against cross site request forgery.

3. TECHNOLOGY STACK

3.1 Application technology Stack

3.1.1 Backend Tier

Name	Type	Purpose
Development Language	Python	Base Development Language for application development
Framework	Django	Python based web application framework
	Django Rest Framework	Django based framework for Restful HTTP resources
Database	PostgreSQL	object-relational database management system
Libraries (Major)	psycopg2	Python-PostgreSQL Database Adapter
	django-cors-headers	Handling the server headers required for Cross-Origin Resource Sharing (CORS)

	chardet	Used to identify encoding of file uploaded from UI and that encoding type is then passed while reading/writing file. Majorly being used to support internalization.
	python-dotenv	Reads the key,value pair from .env file and adds them to environment variable. It is great for managing app settings during development and in production using 12-factor principles.
	boto3	Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) used to connect AWS Services.
Tools	JetBrains PyCharm IDE	IDE used for Python Development
	venv	Module provides support for creating lightweight “virtual environments” with their own site directories, optionally isolated from system site directories.
Application Server	Django Development Server	Django provided development server to test application

3.1.2 Frontend Tier

Name	Type	Purpose
Development Stack	React	The base JavaScript library for Component-based architecture (CBA). Whole frontend is depend on React library.
	Node.js	Scripting language used for core business framework level implementations
	ES6	A standard JavaScript syntax used to make the code maintainable and object oriented
	Axios	A JavaScript library for calling HTTP Request asynchronously, it is used to call APIs from backend server.
	React Bootstrap	Frontend framework based on React
	Yup	A JavaScript object schema validator and object parser – used for fronted UI validations
Packaging & Bundling	NPM	Node Package Manager used for packaging modules and code in one place and managing dependencies conflicts
	Webpack	A Module bundler – it traverses through the source to construct the graph, and it uses this information and configuration to generate bundles
Libraries (Major)	js-file-download	triggers browser to save javascript-generated content to a file
	dotnev	A zero-dependency module that loads environment variables from a .env file into process.env . Storing configuration in the environment separate from code is based on The Twelve-Factor App methodology
Tools	Visual Studio Code	IDE used for development
	Chrome dev tools	For Debugging Node.js and React
	Postman	For validating, testing API Endpoints

3.1.3 Deployment Technology Stack

Name	Type	Purpose
Operating System	Linux - Ubuntu	Host Operating system for deploying application
Application Servers	Nginx	High performance Load balancer, web server used for reverse proxying, caching, load balancing
	Gunicorn	Production grade application server to run Python based applications
Servers Management	Supervisor	Process control system to monitor process in OS. It is used to look after the Gunicorn process and make sure that they are restarted if anything goes wrong, or to ensure the processes are started at boot time.
SSH Access	Putty	terminal emulator application which act as a client for the SSH
	Pageant	A PuTTY authentication agent. It holds private keys in memory so that it can be used whenever connecting to a server.
	PuttyGen	A key generator tool for creating SSH keys for PuTTY. It is analogous to the ssh-keygen tool used in some other SSH implementations
	WinSCP	SFTP client and FTP client for Windows. Copy file between a local computer and remote servers using FTP, FTPS, SCP, SFTP

3.1.4 DevOps Technology Stack

Type	Name	Purpose
Containerization	Docker, Docker Compose	Used for containerization of application into microservices.
Version Control	Git, GitHub	Repository to save project artifacts
Continuous Integration/Continuous Management	Jenkins, Blue Ocean	Automation tool to setup CI/CD process by creating pipelines for code builds and releases.
Monitoring	Prometheus	A monitoring system with a dimensional data model used to monitor and generate metrics for containers and microservices.
Visualization	Grafana	A analytics and visualization tool used for visualizing time series data from Prometheus as a data source.

3.1.5 Infrastructure As Code

Name	Purpose
CloudFormation	AWS Templating service provides provision and management of AWS Services. Used to create complete AWS based architecture by coding the infrastructure configuration into CloudFormation templates. It generated and manages deployment stack by code.
Terraform	A tool for managing and provisioning infrastructure on multiple cloud providers using IaC. It is used in the project to build an automated infrastructure on AWS cloud provider using terraform .tf files.

4. PACKAGING AND BUILDING

4.1 Frontend Tier

Name	Purpose
Babel – JavaScript Transpiler for ES6	
babel-core	Core package for transforming code
babel-cli	allows to compile files from the command line
preset-react and preset-env	env preset allows to transform ES6+ into more traditional javascript and the react preset does the same, but with JSX instead
plugin-proposal-class-properties	Use properties directly on a class
babel-polyfill	includes a polyfill that includes a custom regenerator runtime and core-js. Needed for async & await
Loaders	
babel-loader	Transpile files with Babel and Webpack.
sass-loader	Load SCSS and compile to CSS.
node-sass	Node Sass.
postcss-loader	Process CSS with PostCSS.
cssnano	Optimize and compress PostCSS.
postcss-preset-env	Sensible defaults for PostCSS.
css-loader	Resolves CSS imports into JS.
style-loader	Inject CSS into the DOM.
eslint-loader	Use ESLint with Webpack.
file-loader	Copy files to build folder.
url-loader	Encode and inline files. Falls back to file-loader.
Plugins	
clean-webpack-plugin	Remove/clean build folders.
copy-webpack-plugin	Copy files to build directory.
html-webpack-plugin	Generate HTML files from template.
mini-css-extract-plugin	Extract CSS into separate files.
optimize-css-assets-webpack-plugin	Optimize and minimize CSS assets.
terser-webpack-plugin	Minify JavaScript.
Webpack	
webpack	Module and asset bundler.
webpack-cli	Command line interface for Webpack.
webpack-dev-server	Development server for Webpack.
webpack-merge	Simplify development/production configuration
cross-env	Cross platform configuration.

4.2 Backend Tier

All dependencies of Backend tier is bundled by pip tool in requirements.txt. Freeze command of pip is used to collect all dependencies of application into requirements.txt.

“Requirements files” are files containing a list of items to be installed using pip install. Logically, a Requirements file is just a list of pip install arguments placed in a file.

5. DEPLOYMENT MANAGEMENT

5.1 Local Servers Setup

2 Local servers are setup for application deployment to test and verify the project artifacts.

- Dev Server
- QA Server

Both servers are running on separate Virtual Machines on Linux Operating system.

All Development work or enhancements are fist tested locally on Development machine and then QA Machine. Once the build artifacts passed all test cases, it becomes ready to move into PROD servers running on Cloud.

5.2 Version Controlling

GitHub Version Controlling system is used in this project as a code repository. Project artifacts are saved in 'Private' Repository of GitHub. All code is first committed to 'Development' Branch and then successful testing it get merged into 'Master' branch.

5.3 Containerization

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Docker is used as a containerization mechanism to create executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings. It provides the ability to package and run an application in a loosely isolated environment.

Application containerization is achieved in 2 formats:

- Via standalone containers
- Via Docker Compose

Best Practices followed in Docker Containerization:

✓ Tried to reduce the number of layers in all docker images by minimizing the number of separate RUN commands in the DockerFiles
✓ Avoided storing application data in container's writable layer
✓ Creating ephemeral containers
✓ Used restart policies to control whether containers start automatically when they exit, or when Docker restarts. Restart policies ensure that linked containers are started in the correct order.

5.3.1 Via standalone containers

Each tier of application is deployed in isolated container which is self-managed and loosely coupled from other tiers.

3 containers are used for application deployment :

- Web Container – contains frontend tier consisting Nginx server
- API Container – contains backend tier consisting Python Gunicorn server

- Database Container – contains data store tier consisting PostgreSQL server

5.3.2 Via Docker Compose

In this case, application is deployed in 3 isolated containers in the same way as mentioned above but all are controlled by Compose. That means, managing containers – starting, stopping – all is controlled by compose itself. All containers can up and down by single command of Compose.

5.3.3 Docker Volumes

Volumes from Docker containers are mounted on Host machine to access the logs and other files generated inside the containers.

Web Container Volumes	**Codebase volume** Type: Bind mount Used to mount the codebase folder of host to docker container, so that file changes can directly applied to the container
	Log Volume Type: Volume Nginx logs mounting
API Container Volumes	**Codebase volume** Type: Bind mount Used to mount the codebase folder of host to docker container, so that file changes can directly applied to the container
	Log Volume Type: Volume Translator application logs mounting
Database Container Volumes	**Data Dictionary Volume** Type: Named Volume Named volumes are stored inside the Docker Daemon directory with the name of named volume. Name is prepended by the project name by compose.
	Log Volume Type: Volume PostgreSQL log mounting

5.3.4 Docker Networking

Docker containers will be running in different instances in different subnets, they cannot communicate each other by using 'Bridge' network as it only works within same machine. In this application each container will be running in different machine in multiple docker daemon hosts. To communicate standalone containers in different machines, 'overlay' networking is used.

The overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate securely.

To enable the communication between all containers using overlay network, Docker Swarm is used as a cluster management and orchestration took. API Instance acts as a ‘manager’ node and remaining all other nodes work as a ‘worker’ nodes.

5.4 Continuous Integration/Continuous Management

Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results etc.

Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button.

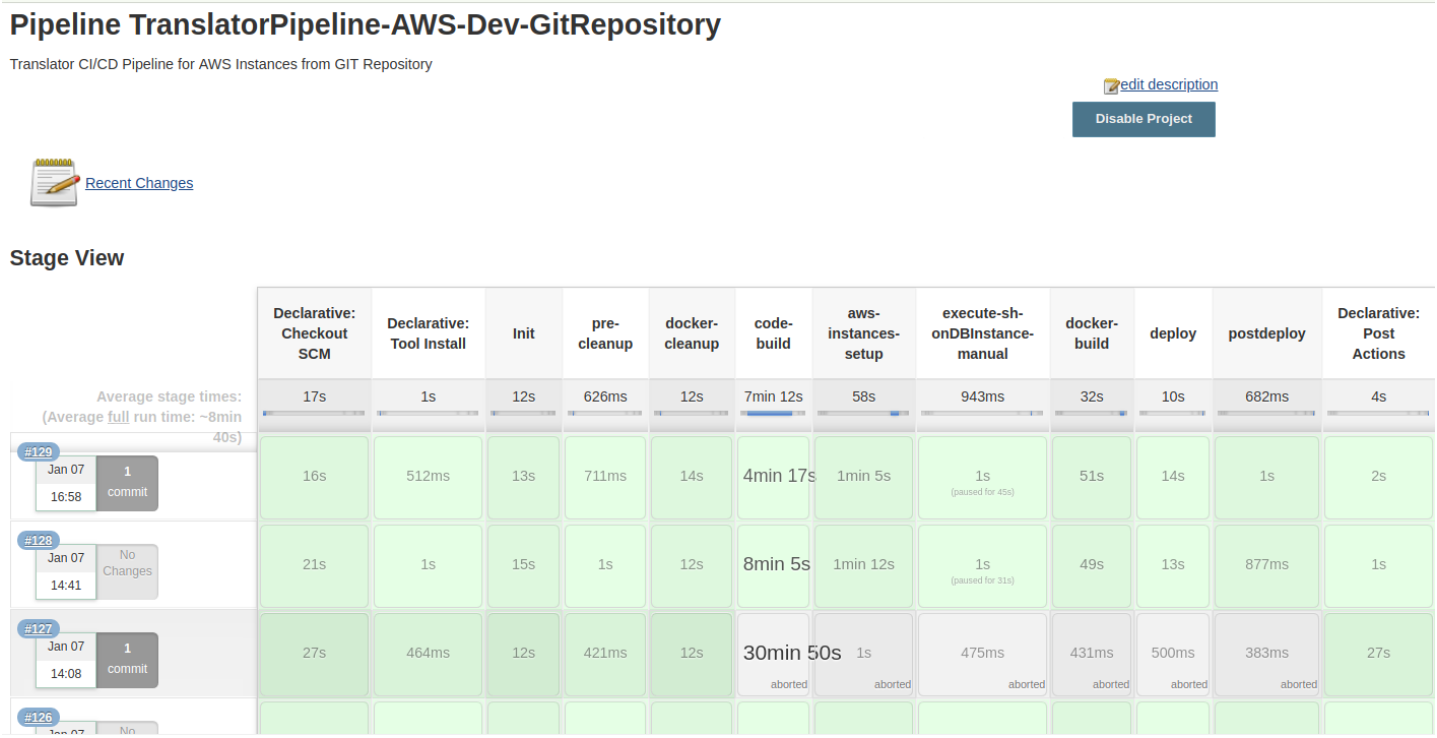
Jenkins tool is used for all CI/CD pipelines development.

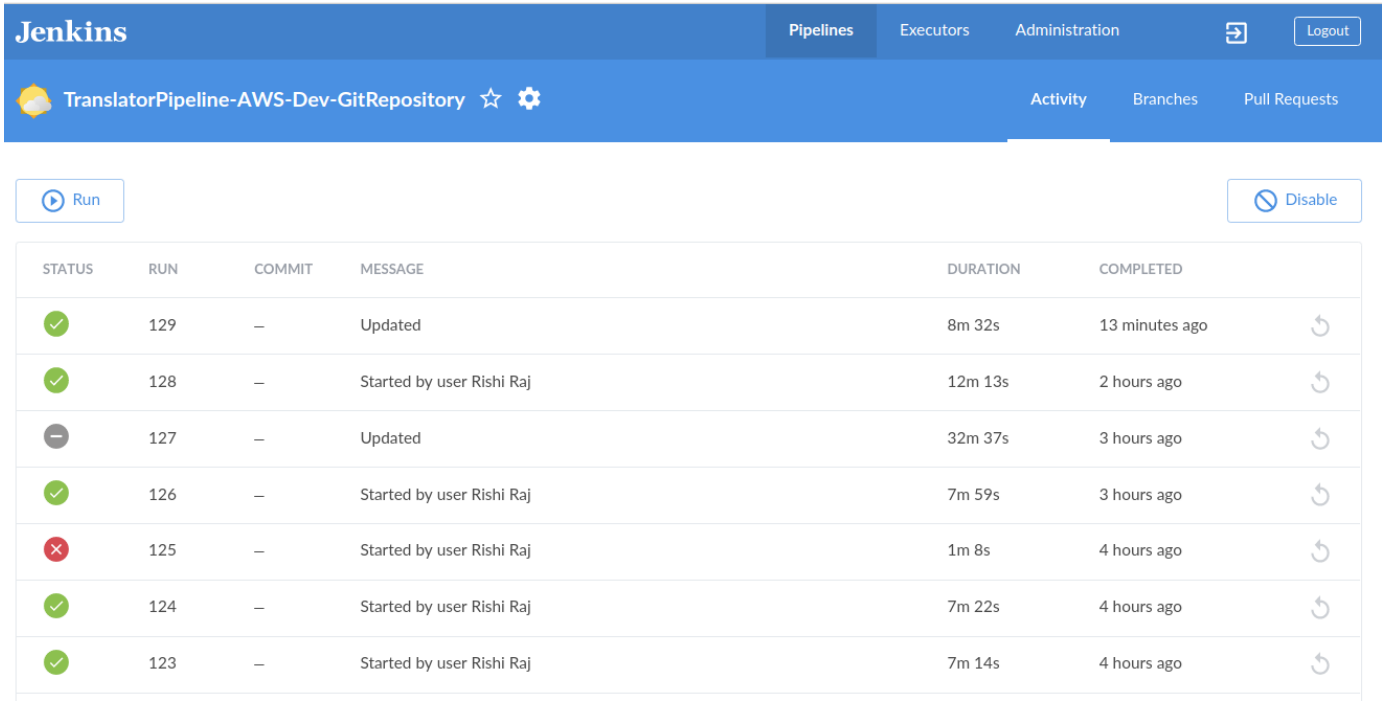
- Node.js and GitHub integration with the required plugins is setup in the Jenkins to support project build requirements.
- SCM based GitHub repository is used to get the source code and build the application
- AWS Authentication details are setup in Jenkins credentials system instead of using in the Jenkinsfile

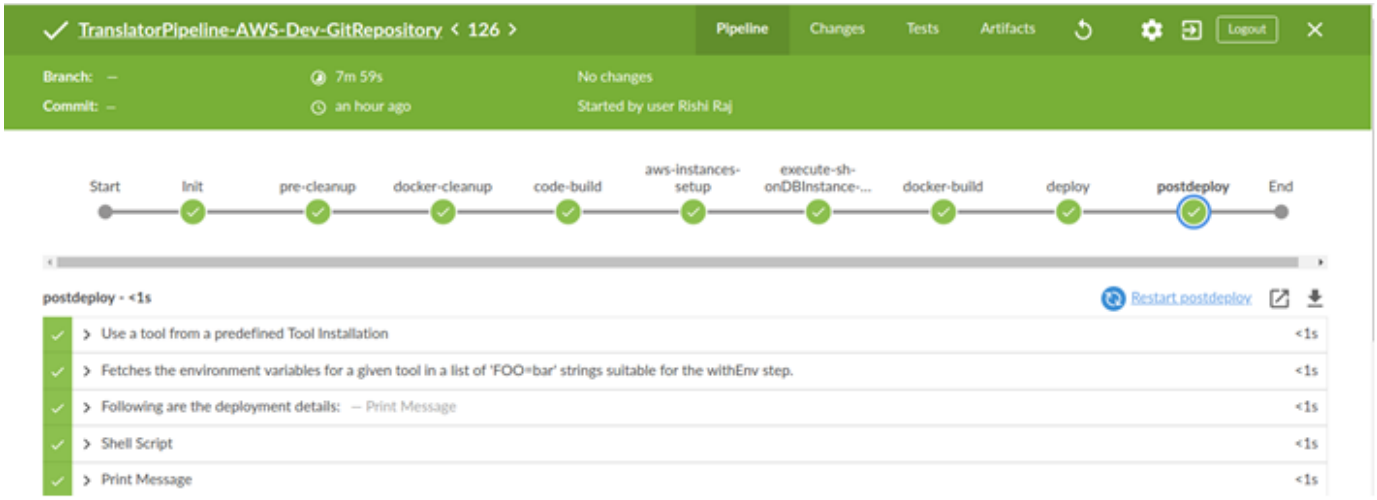
2 set of Jenkins pipelines are developed:

- Pipeline for Local Development and QA servers
- Pipeline for AWS Cloud Deployment

Following are the few shots from Jenkins Pipeline executions and Blue Ocean:





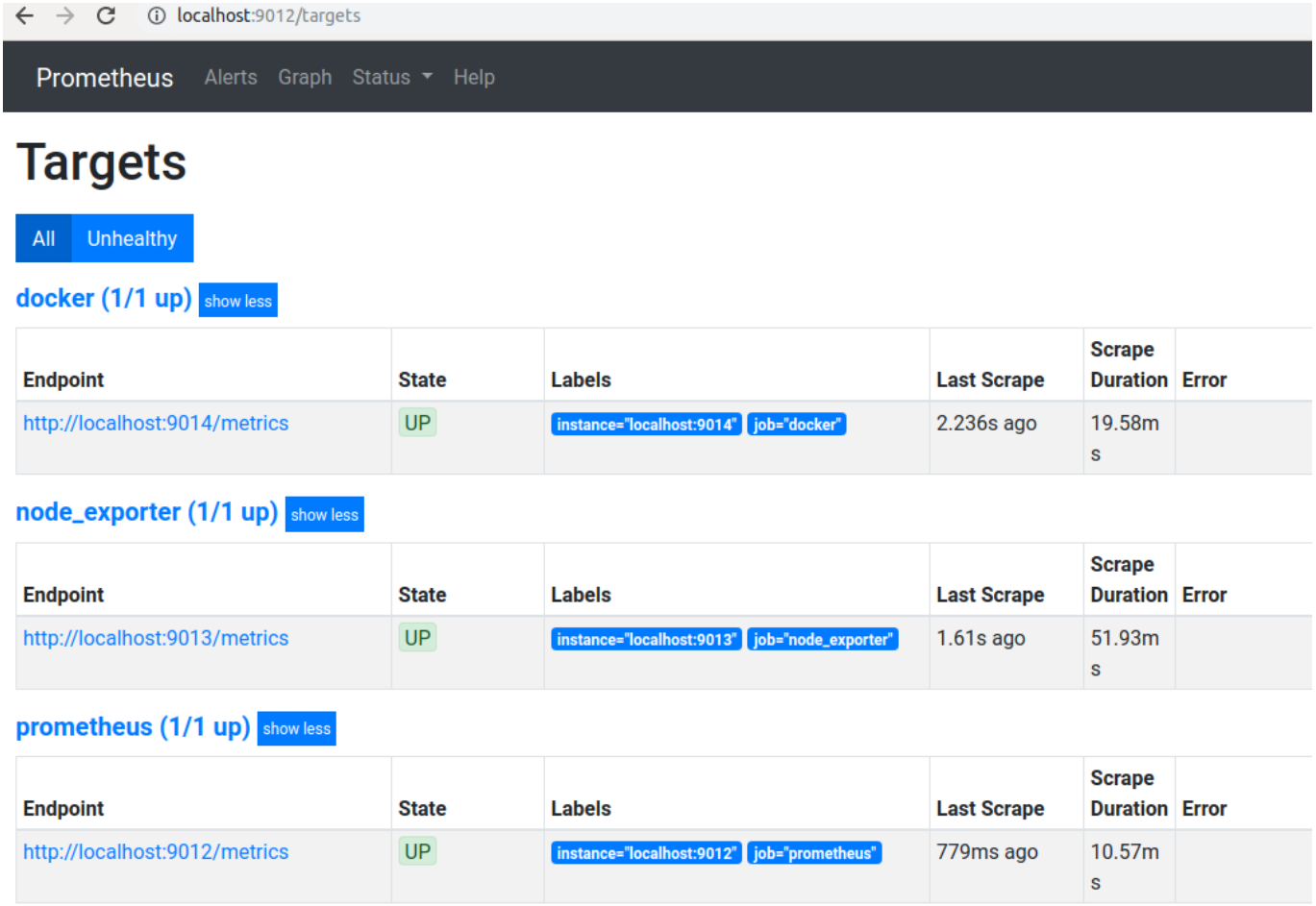


5.5 Monitoring using Prometheus

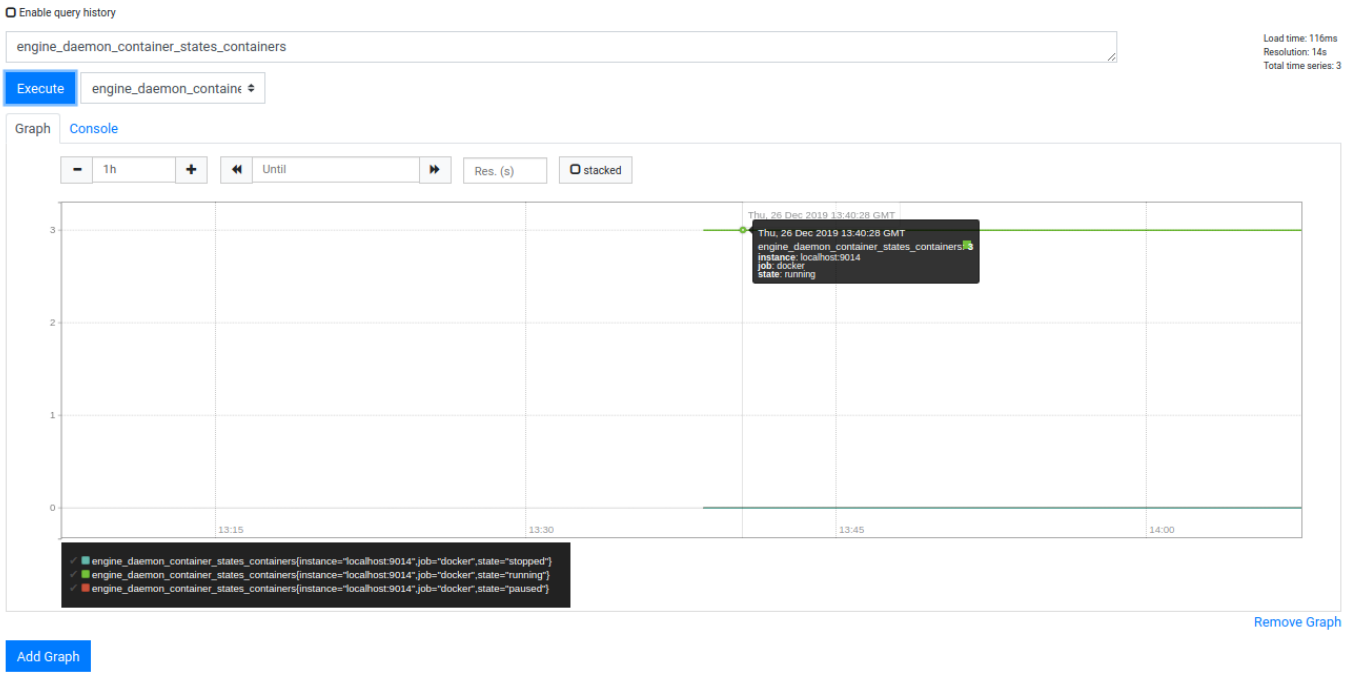
Prometheus is used for monitoring Docker activities. Prometheus is a monitoring platform that collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets.

Docker is configured as a Prometheus target. Prometheus is configured to add a job to monitor Docker and generate metrics.

Following are the few shots from Prometheus monitoring Docker target :


localhost:9012/targets
Prometheus Alerts Graph Status Help
Targets
All Unhealthy
docker (1/1 up) show less
Endpoint State Labels Last Scrape Scrape Duration Error
http://localhost:9014/metrics UP instance="localhost:9014" job="docker" 2.236s ago 19.58ms
node_exporter (1/1 up) show less
Endpoint State Labels Last Scrape Scrape Duration Error
http://localhost:9013/metrics UP instance="localhost:9013" job="node_exporter" 1.61s ago 51.93ms
prometheus (1/1 up) show less
Endpoint State Labels Last Scrape Scrape Duration Error
http://localhost:9012/metrics UP instance="localhost:9012" job="prometheus" 779ms ago 10.57ms


Enable query history
engine_daemon_container_actions_seconds_sum
Load time: 163ms
Resolution: 14s
Total time series: 5
Execute
Graph
Console
1h
Until
Res. (s)
stacked
25
20
15
10
5
0
13:15
13:30
13:45
14:00
engine_daemon_container_actions_seconds_sum{action="start",instance="localhost:9014",job="docker"}
engine_daemon_container_actions_seconds_sum{action="delete",instance="localhost:9014",job="docker"}
engine_daemon_container_actions_seconds_sum{action="create",instance="localhost:9014",job="docker"}
engine_daemon_container_actions_seconds_sum{action="commit",instance="localhost:9014",job="docker"}
engine_daemon_container_actions_seconds_sum{action="changes",instance="localhost:9014",job="docker"}
Remove Graph
Add Graph



5.6 Visualization using Grafana

Grafana is a graph and dashboard builder visualizing time series infrastructure and application metrics. It allows to create alerts, notifications, and ad-hoc filters for the data.

Prometheus added as a data source to fetch the data of Docker target into Grafana.

6. INFRASTRUCTURE AS CODE

All infrastructure components on AWS cloud are defined and configured using IaC. CloudFormation and Terraform IaC tools are used to create infrastructure on AWS.

Following infrastructure components are created as a part of IaC development in AWS Cloud.

6.1 Using CloudFormation

CloudFormation service is implemented to create automated infra configurations, it uses template (Blueprint) to define the infra structure on AWS.

- To update Deployment stack after creation 'Changesets' mechanism of CloudFormation is used to incrementally update the infrastructure stack on AWS.
- One Click Launch Deployment link is created with all the Deployment configuration in-built to quickly create all infrastructure components on any AWS account with single click.

6.2 Using Terraform

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently.

- Isolated **workspaces** are used to store the settings for infrastructure deployment separately for each different environment like Dev, QA, Prod.
- **Modules** approach of Terraform is used to structure and organize the infrastructure components. A module is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that the infrastructure can be defined in terms of its architecture, rather than directly in terms of physical objects.
- **AWS Authentication** – To keep the AWS authentication keys secure and not to explicitly define in the code, authentication with AWS is done based on the ~/.aws/credentials file which gives advantage to avoid sharing AWS access keys directly in the code. Profile from that credentials file is used to authenticate with AWS. This authentication is based on IAM user.
- **AWS Provider versioning** - Each time a new provider is added to configuration -- either explicitly via a provider block or by adding a resource from that provider -- Terraform initializes the provider before it can be used. When terraform init is re-run with providers already installed, it will use an already-installed provider that meets the constraints in preference to downloading a new version. Provider versioning helps in avoiding version conflicts.
- **Execution Plans** are monitored to inspect the infrastructure changes being done by Terraform before applying any changes on AWS

7. AWS CLOUD SETUP

7.1 Infrastructure Bifurcation

<table>
<tr><th>Name</th><th>Purpose</th></tr>
<tr><td>EC2 Instances</td><td>3 Public Instances:
Web Server (Nginx) instance
API Server (Python) Instance
NAT Instance
1 Private Instance:
Database server (PostgreSQL) instance</td></tr>
<tr><td>EBS</td><td><table><tr><td>Volume Type</td><td>General Purpose SSD (gp2)</td></tr><tr><td>Size</td><td>20 GiB</td></tr><tr><td>Availability Zone</td><td>us-east-1a</td></tr></table></td></tr>
<tr><td>ELB</td><td>Application Load Balancer is configured</td></tr>
<tr><td>VPC</td><td>Cidr block: 10.11.0.0/16</td></tr>
<tr><td>Subnets</td><td>2 Public Subnets
Web Subnet
Used for Web and NAT instance
Cidr: 10.11.1.0/24
API Subnet
Used for API Instance
Cidr: 10.11.2.0/24</td></tr>
</table>

	1 Private Subnet • DB Subnet Used for DB Instance Cidr: 10.11.3.0/24
Route Tables	Route table 1 • For Public Subnet • Connected to igw Route table 2 • For private Subnet • Connected to NAT Instance
Internet Gateway	IG connected to VPC
Network ACLs	Default Network ACLs on subnets
Security Groups	4 Security Groups for each 4 instance
S3	S3 buckets for CloudFormation templates and application user uploads
Key Pair	Generated to access EC2 instances via SSH
IAM	IAM Group IAM Role IAM USer

7.2 Bootstrapping EC2 Instances

User Data Scripts and CloudFormation Helper Scripts (Cloud Init, CFN Signals) are handled on EC2 instance bootstrap to load and setup the instances for the application deployment. All system level initialization to prepare the machine for running the application successfully are coded in User Data of instance. Shell scripts are used to define the setup instructions.

7.3 Setup Specifications

The configuration for this application includes a virtual private cloud (VPC) with 3 public subnets and one private subnet.

The instances in the public subnet can send outbound traffic directly to the Internet, while the instances in the private subnet cannot. Instead, the instances in the private subnet can access the Internet by using a network address translation (NAT) instance that resides in public subnet. The database servers can connect to the Internet using the NAT instance, but the Internet cannot establish connections to the database servers.

Following are the configuration details structured in AWS:

1. AWS **Key pair** is created for to access EC2 instances from SSH.

2. **Secure IAM Setup to AWS services** – IAM service is setup to administrator securely control access to AWS resources. Following are configured inside IAM service for this application:

   - **IAM User** - An IAM user is an identity within the AWS account that has specific permissions for a single person or application.
   - **IAM Group** - An IAM group is an identity that specifies a collection of IAM users.
   - **IAM Role** - An IAM role is an identity within the AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person.

3. A **VPC** with a size /16 IPv4 CIDR block (example: 10.11.0.0/16). This provides 65,536 private IPv4 addresses.

4. Three **public subnets** are defined with a size /24 IPv4 CIDR block. This provides 256 private IPv4 addresses. *A public subnet is a subnet that's associated with a route table that has a route to an Internet gateway.*

Web Server Instance Subnet Contains NAT Instance	10.11.1.0/24
API Server Instance Subnet	10.11.2.0/24

5. A **private subnet** with a size /24 IPv4 CIDR block. This provides 256 private IPv4 addresses.

Database Server Instance Subnet	10.11.3.0/24

6. An **Internet gateway**. This connects the VPC to the Internet and to other AWS services.

7. A **custom route table associated with the public subnet**. This route table contains an entry that enables instances in the subnet to communicate with other instances in the VPC over IPv4, and an entry that enables instances in the subnet to communicate directly with the Internet over IPv4 via Internet Gateway.

Destination	Target
10.11.0.0/16	local
0.0.0.0/0	igw-id

8. A **custom route table associated with the private subnet**. This route table contains an entry that enables instances in the subnet to communicate with other instances in the VPC over IPv4, and an entry that enables instance in the subnet to communicate with NAT instance to connect with the Internet over IPv4 for software installs and updates.

Destination	Target
10.11.0.0/16	local
0.0.0.0/0	nat-instance-id

9. Four Security Groups are configured which acts as a virtual firewall for instances to control inbound and outbound traffic. One Security Group for each instance. Ports for Docker overlay networking are also opened here along with HTTP/s, SSH and PostgreSQL ports.

10. Three Instances in the public subnet with Elastic IPv4 addresses (example: 198.51.100.1), which are public IPv4 addresses that enable them to be reached from the Internet. The instances can have public IP addresses assigned at launch instead of Elastic IP addresses. Instances in the private subnet are back-end servers that don't need to accept incoming traffic from the Internet and therefore do not have public IP addresses; however, they can send requests to the Internet using the NAT gateway. Dedicated IAM Role is defined for these instances.

    There instances are:
    - Web server Instance – Containing Nginx server
    - API Server Instance – Containing Python Gunicorn Server
    - NAT Instance – To connect database instance to internet

11. Single Instance in the private subnet with private IPv4 addresses. This enables them to communicate with each other and other instances in the VPC. This instance can communicate with the Internet only from NAT Instance.

12. S3 buckets of ‘Standard’ type are created to store CloudFormation templates and application user uploads

7.4 Monitoring EC2 Instances

EC2 instances are being monitored by using following possible ways:

- System Status Checks
- Instance Status Checks
- *CloudWatch Alarms* – Required metrics are defined in CloudWatch to monitor EC2 instances, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods.
- *CloudWatch Events* – to automate AWS services and respond automatically to system events. Events from AWS services are delivered to CloudWatch Events in near real time, and automated actions are specified to take when an event matches a rule.
- *CloudWatch Logs* - monitor, store, and access log files from Amazon EC2 instances, AWS CloudTrail, or other sources.

8. AWS COST MANAGEMENT

8.1 Billing and Costing Alerts

Billing and Costing alerts are setup to monitor the costs.

- **Cost Budget** is created with the threshold value that will automatically send alert when the AWS costs or usage exceed, or are forecasted to exceed, the thresholds set.

- **CloudWatch Alarms** created for the appropriate metrics that can send email when the threshold is about to expire, can be set based on the service. For example, a CloudWatch alarm can be set for 'EstimatedCharges'

- **AWS Cost & Usage Reports** – One report in this section is created that tracks the AWS usage and provides estimated charges associated with the AWS account. The report contains line items for each unique combination of AWS product, usage type, and operation that AWS account uses.

8.2 Monitoring Costs

Manual monitoring is being done by observing following cost management AWS Services:

- Billing & Cost Management Dashboard
- Cost Explorer
- Bills