

HealthCare Project

Patient-Care

Overview

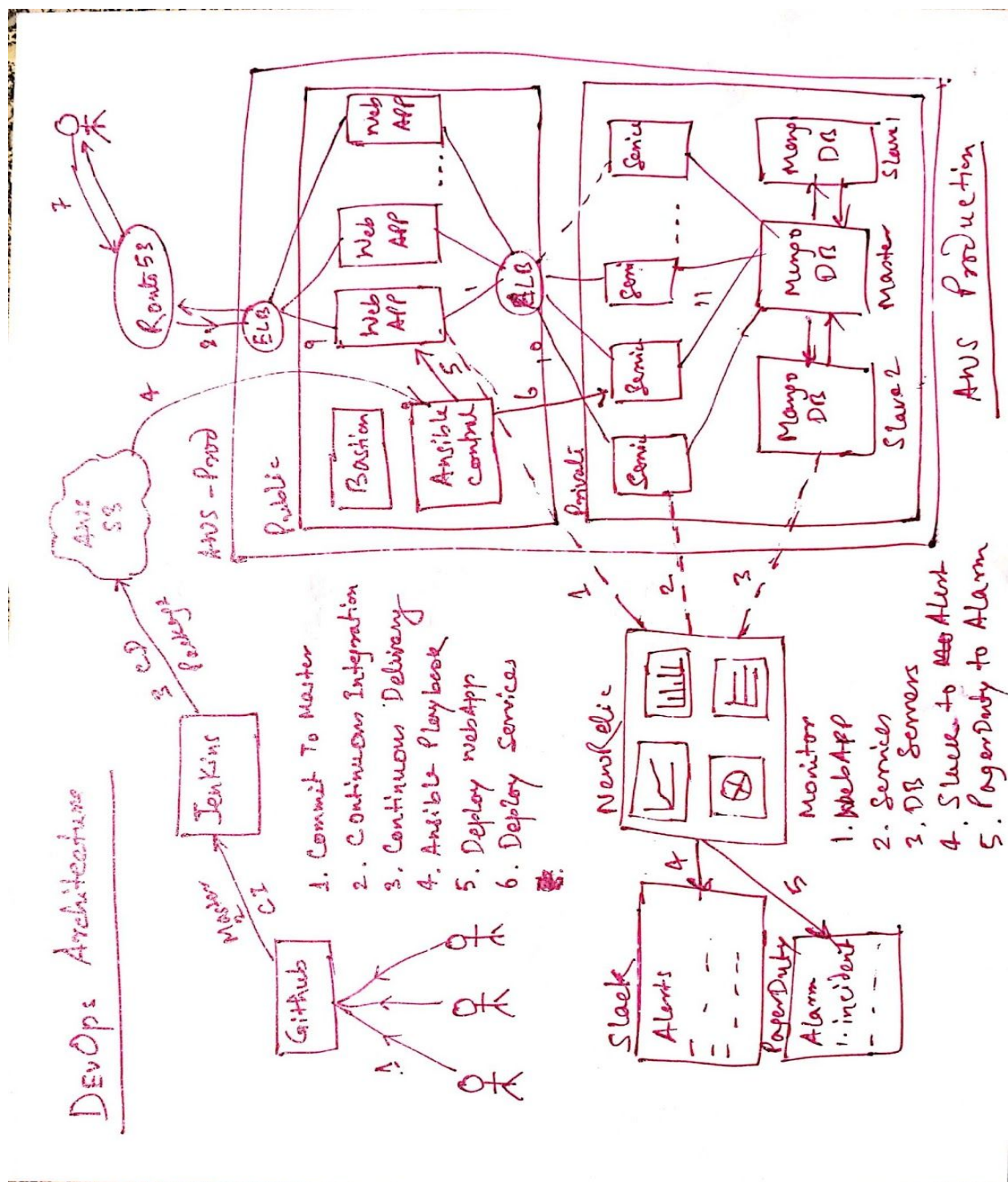
Patient-Care is a system which manages patient master data with the following components hosted on AWS Public Cloud System

- A API/Service layer which exposes CRUD operations for a patient backed by a SQL database.
- A Message bus where all changes to patient changes are posted for downstream consumers to consume.
- A UI using which approved personal can approve changes done to patients. This UI uses a Node server for its APIs.
- An offline background system which reads patient records (every night) and does analysis like find-duplicates etc.

As a DevOps project, the following aspects of the Deployment and Hosting.

- Design & Setup CI-CD pipeline - GitHub & Jenkins
- Develop a deployment architecture assuming the deployment environment is AWS - Staging & Production
- How is environment setups automated in such a system - Packer & Terraform
- What would be the CD pipeline for such a system to deploy code bits - Ansible
- How update & upgrade packages - Ansible
- What components are needed to monitor and debug such a system - NewRelic/DataDog/Nagios
- How can such a system protected against DDOS attacks - Bastion Jumphost & Patching
- How can we achieve zero-downtime deployments in such a system - Blue/Green Deployment or Canary Deployment
- Data Backup & Disaster-recovery on such a system - AWS S3 CLI & Cronjob
- Log Management - Splunk
- Alert & Alarm system issues - Slack & PagerDuty

DevOps Project Workflow :



CI-CD Pipeline (GitHub + Jenkins):

GitHub:

1. GitHub as SCM
2. Jenkins Server Setup
3. Jenkins Jobs or Pipeline for CI
4. Uploading package to Repository(AWS S3) for Continuous Delivery

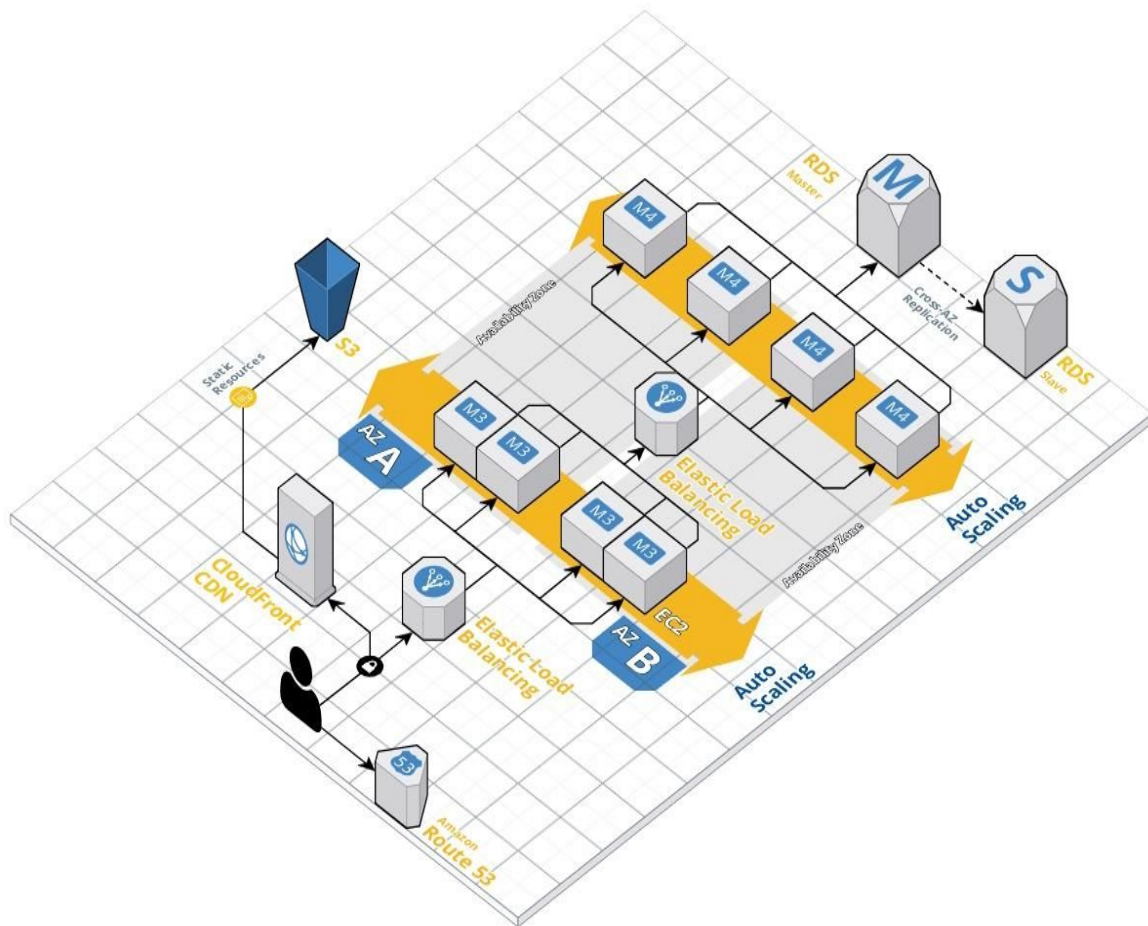
Configuration Management (Ansible):

GitHub Repo : <https://github.com/devops-ccube/demo-project/tree/master/ansible-cfm>

Ansible Roles :

1. SSH Key - Configure SSH keys for accessing VM's from Bastion Server only
2. Ansible - Setting up Ansible-Control Machine for executing ansible playbook to Deploy Web, Rest API & DBServer to their respective VM's
3. NodeJs - NodeJS & NPM Software installation & Configuration for nodejs based application.
4. NewRelic Agent - NewRelic Agent Roles will install NewRelic Agent on all servers which are critical & needs monitor system metrics
5. Deploy Package - Ansible Playbook to Download packages from AWS S3 repository and Deploy Web & Rest API on WebServers & Backend Servers respectively.

Infrastructure Architecture/Build (Terraform + Packer):



Packer Code: Build Automated AMI

Github Repo : <https://github.com/devops-ccube/demo-project/tree/master/packer-ami>

Pre-Baked Base AMI Prerequisite : ssh keypair(id_rsa, authorized_keys)

- Bastion Server(Jumphost) AMI
- Ansible Control Machine AMI
- Services AMI
- WebApp AMI
- Database AMI
- Splunk Server AMI
- Jenkins Server for CI/CD

Steps to Build AMI :

- Login to any Linux Machine
- Clone the code to Linux Machine
- Install & Configure Packer
- Install Ansible & Python2.7
- Configure AWS Access keys
- Go to folder “packer-ami”
- Execute command to build Bastion AMI : packer build bastion-ami.json
- Execute command to build WebApp AMI : packer build web-ami.json
- Execute command to build Ansible Control AMI : packer build ansible-control-ami.json
- Execute command to build MongoDB Server AMI : packer build mongodb-ami.json

Terraform Code:

GitHub : <https://github.com/devops-ccube/demo-project>

1. Clone the Repository
2. Change to “terraform-infra” folder
3. Configure your AWS Keys(Administrative Access)
4. Change to folder environment “production” or “staging”
5. Create AWS S3 folder for S3 backend
6. Run : \$terraform init (Initialize terraform code & download aws libraries)

7. Run : \$terraform plan (Show infrastructure to build)
8. Run : \$terraform apply (Create Infrastructure)
9. Run : \$terraform destroy (Destroy Infrastructure within 1 hour to save bill)





Folder environments

This folder has one folder for each environment, in this case it has production and staging, but it could have as many as you need! For example if you want each developer to have its own environment they could copy one of those folders and have one new environment, as easy as it sounds!

Each folder has some files that basically wrap the contents of the modules folder.

Folder modules

Here we define all our infrastructure leaving some open parameters for customizing each environment, for example in production you want to run some powerful machines, but for staging you might want low power machines to reduce costs.

This folder has two folders, state which is a module for creating the required resources for managing terraform state remotely, an S3 bucket and a DynamoDB table, which we use in each environment. This is one possibility of the many available. The second folder infrastructure contains the definition of all the infrastructure which we use in each environment.

Folder shared

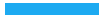
This folder is for throwing some shared files. In order to use the files we simply create a symbolic link wherever we need to use its contents.

Modules

Terraform allows you to create and use modules. That enables us to reuse code.

State module

We have a state module which we use to encapsulate the resources for our backend of choice: S3 Bucket for the state file and DynamoDB for state locking.



The reason to have the state module independent of the infrastructure module is to give us more flexibility, for example, if you want to have one environment per developer they don't need remote storage or locking, they can simply use a local state file.

Infrastructure module

Here I have used the module idea to create an infrastructure module that represents everything that is needed to run our apps, that means that every time you use that module you get an exact copy of your infrastructure.

This module is only glue for the modules it includes, in the modules/environment/main.tf file we glue the network, instances and databases modules.

Network module

It represents the "cables" of our infrastructure, it only has one variable that is the name of the environment.

It outputs some other variables like the CIDR Blocks of the network, the ID of the VPC, and IDs of the public and private subnets.

Databases module

This module receives as input the current environment, the IDs of the private subnets and creates the required resources for a Multi AZ highly available database instance.

After creating the module it exports the endpoint of the database and some other data.

Instances module

The instances module represents the VMs that will be running the code, We also include an AWS Application Load Balancer



Code Deployment :

1. SSH to Ansible Control machine.
2. Checkout Ansible github code
3. Run Ansible Playbook to Deploy Packages.

System & Application Monitoring :

1. NewRelic Agent is installed on all servers by default from AMI's.
2. Login to your NewRelic Dashboard account & Check Infrastructure sections for System Metrics
3. Integrate NewRelic with Slack using keys
4. Integrate NewRelic with PagerDuty for Alarm & Escalation as per scheduled

System Troubleshooting :

1. High CPU Utilization : Check which process is taking more CPU, restart that services or thread
2. High Memory Utilization : Check which Service is taking more Memory, kill that service and restart again to release memory
3. Disk Full : Check which disk is full & which is taking more space like backup or log file. Do backup remotely & cleanup or archive.
4. Portal is Down : Check all your deployed application status, if any services or application are down. Start the application or service.

Centralized Logging System : Splunk/ELK

1. Splunk forwarder Agent is installed on all servers by default from AMI's
2. Config file of Splunk forwarder agent points to Splunk Server for sending logs
3. All logs are collected to Splunk Server from all application & services server's
4. Splunk Console will provide access to analyse logs & debug issues at any point of time.

DevOps: Roles & Responsibility -

- Designed and implemented scalable, secure cloud architecture based on Amazon Web Services.
- Implemented Automated **Build** and **Deployment** process for applications and re-engineering set up for better user experience and leading up to building a **Continuous Integration/Delivery** System.
- Maintain and Enhance the Enterprise DevOps Platform tools, services and software's like **Jenkins, Git/GitHub, Maven, SonarQube, Artifactory, Terraform, Packer, Ansible, Splunk, NewRelic & PagerDuty** etc.
- Involved in Infrastructure **Orchestration** of **Staging & Production** Environment on **AWS** Cloud using **Packer, Terraform & Ansible**.
- Involved in Infrastructure **Design & Documentation**.
- Worked with **AWS** related activities like **Backup, Monitoring, Alerting**, Security Patches, keep an eye on **AWS Costs** and take corrective actions.
- Involved **Release & Deployment** process in **Staging & Production** Environment using **Blue-Green Deployment**.
- Implemented **Continuous Integration (CI) and Continuous Delivery (CD)** using **Jenkins** CI which has very strong build pipeline consists of Build verification, Junit tests, Deployment Tests, API tests, Service Tests and Functional Tests.
- Used **Jenkins+Github** and **MAVEN** for building and delivering application artifacts.
- Implemented **Ansible** for Configuration Management to deploy into different applications into Staging & Production Environment.
- Automating Redundant tasks using **Shell** Scripting.
- Performed Branching, Tagging, Release Activities using Version Control Tools GIT and GitHub.
- Performed and deployed builds for various environments like **Dev, Test, QA, Staging and Production Environments**.
- Using **SonarQube** for Continuous Code Quality to attain the Security for Health Industry.
- Involved in **Monitoring** the **Servers** and **Applications** using **NewRelic** and **Splunk**.
- Involved in **troubleshooting** production server issues in day to day activity



AWS: Roles & Responsibility -

- Designed and implemented scalable, secure cloud architecture based on Amazon Web Services.
- Extensively worked on moving existing on-premises data to AWS cloud storage.
- Maintained and provisioned EC2 instances, S3, IAM, VPC, Route53, ELB, Redshift, CloudWatch, SNS, SQS, AWS CLI.
- Implemented cost optimization across different application architectures using EC2 reserved/spot instances and utilize EBS to store persistent data and mitigate failure by using snapshots.
- Created multiple IAM users and groups to secure AWS resources only for personnel authorized to use those resources.
- Creating S3 buckets and maintained and utilized the policy management of S3 buckets and Glacier for storage and backup on AWS.
- Designed AWS CloudFormation templates to create custom sized VPC, subnets, NAT to ensure successful deployment of Web applications and database templates.
- Utilize CloudWatch to monitor EC2 instances, DynamoDB tables, CPU memory and create an alarm for notification or automated action for better understanding and operation of system.
- Implemented Route 53 DNS service in effectively coordinating the load balancing, fail-over and scaling functions.



Interview Question & Answers -

- Design & Setup CI-CD pipeline - GitHub & Jenkins
- Develop a deployment architecture assuming the deployment environment is AWS - Staging & Production
- How is environment setups automated in such a system - Packer & Terraform
- What would be the CD pipeline for such a system to deploy code bits - Ansible
- How update & upgrade packages - Ansible
- What components are needed to monitor and debug such a system - NewRelic
- How can such a system protected against DDOS attacks - Bastion Jumphost & Patching
- How can we achieve zero-downtime deployments in such a system - Blue/Green Deployment
- Data Backup & Disaster-recovery on such a system - AWS S3 & Cronjob
- Log Management - Splunk
- Alert & Alarm system issues - Slack & PagerDuty

