Christy Henitha JV
Technizant

eGov – Real time implementation

Technology

(Given below applies for most of the egov projects.)

Development & Deployment: Devops

Architecture: Microservices / Monolithic / CoEx stack

Operating System (for application development): Windows, **Ubuntu Linux** (big size docker image), **Alphine Linux** (comparatively smaller size docker image) - Favourable OS for creating docker containers.

Operating System (for deployment): Any operating system, since microservices run as docker container, capable of running in any OS.

Front end: Nodejs / angular

Back end: Java Springboot

Database: PostgresSQL

Source code repository: Github (Important concepts: push, pull)

Source code: github.com\egovernments\egov-services

Cloud computing: AWS (Amazon Web services), Azure (Microsoft)

Build tool: Maven (Important concept: Build now)

CI /CD: Jenkin (Important concept: pipleline, NPM build, maven build, deploy to dev)

API gateway: Ngnix, Zuul

Containerization: Docker

Orchestration: Kubernetes - KOPS (Kubernetes Operation), EKS (Elastic Kubernetes Service) / AKS (Azur Kubernetes Service). Important concepts: Ingress, configmap, secrets, volume, cluster, node, pod

Roles

Mandatory:

Devops Engg - Microservices

Devops Engg - ERP Monolithilic

Optional:

Team leader

Devops Mentor

Scrum master

Activities of a team leader

- No specific role for team lead currently. Understand roles of devops engineer & then track his progress.
- A team leader should be knowing build & deployment lifecycle.
- If any microservice functioanlity has to be created using new technology (apart from Java springboot & NodeJS), team leader has to give strategy how to build and deploy new technology service in kubernetes environment.
- Encourage Devops Engineer for creating best practices, to improve the current process / tool.
- Co-ordination or involvement in giving demo to Govt authorities with core modules like access control, authorization, session creation & mobile app. Then we can ask for customization for the existing modules and get inputs for new modules required.

Daily activities of Devops Engg:

- Work from home option, not preferred for devops engg, since he/she has to be with dev team, expecting a build at any time & have to be available always online.
- The client will give their AWS or azure or bare metal account, which is the environment, they want to deploy cluster. Dev & QA team has complete access to all modules. UAT & production environment managed for client will have only required modules.
- A devops engg may not have complete visibility of requirements shared by govt. Product team & Dev team are actively involved in meetings to understand new requirements or change in existing requirements.
- For eGov projects, appearance is more important than functionality. Each Govt has unique content for their home page / User Interface. Any defect in appearance will get notified first, before anyone can find fault with functionality.
- JIRA is used for major issues, not solved for a long time, which needs regular follow-up. It is also used for posting new tickets for innovative ideas.
- Google hangouts is used for internal communication among project team.
- All developers may not be knowing devops setup. They know to build and test locally. Devops engg must be able to guide them to create build in the proper required way, so that it can be easily deployed later.
- Automation Tools + Devops Process knowledge is important in Devops.
- While handling tools, we should also clearly know, what are the default values provided by tool and which values should be changed as per project requirement.
- Devops engg should **create correct CI/CD pipleline** in Jenkins and make sure deployment is done in right environment (CI/CD can be done as freestyle also. But we prepare pipleline, i.e. break up of different stages, so that trouble shooting in case of issues, will be easier.)
- Main job is to receive build from dev team and **deploy** to required environment like **QA or UAT or production**, followed by verification of deployment.
- Defects with different levels of severity will be found in pre-production & production, which will be handled by QA team. Once the build is QA approved, it is deployed to production / live environment.
- Whenever, there is an issue in deployment, devops engg will support in trouble shooting.

- After build passes UAT, it will be deployed to production. Deployment to production is easy, since we already know the list of builds to be deployed. We have to just copy build id & paste in Jenkins, it will get deployed automatically, followed by **verification of deployment.**
- For example, if the newly added microservice is not working, we have to check why container didn't start, using **logs** & troubleshoot.
- Monitoring & measure the performance of application using tools like **Kibana**
- **Troubleshooting after deployment is crucial** when compared to other activities. If the container is not running, devops engg should take logs and give to QA team, since QA team has access only to build, not to the environment.
- For each release, devops engineer can make a note of the commit id in GIT, for troubleshooting in later stage.

Additional expectations than day to day activities:

Technical readiness - able to adapt to new technology

Innovative ideas - to improve current process and introduce new tool to reduce time, cost & improve efficiency (value additions)

Devops Engineer - Project scope

- SysOps and DevOps skills to support eGov's Digit, ERP and CoEx stacks. (Microservices & Monolith)
- Per env/per state ownership model to own Day2Day, Release & Deployments.
- Incrementally expand capacity and ownership. Current forecast is for 9 - 12 months. Quarterly review and revise.
- Every env specific tasks are assigned through Tickets and to meet the SLAs based on the Priority.
- Everyday scrum and weekly status call to track the progress.
- Engagement starts with initial KT in eGov office and then should be managed remotely.
- Escalation POC is responsible to address the deviations, schedule variance, capacity, status reports, etc.

Skills Required for MicServices: Understanding on the Microservice architecture, Dockers, Kubernetes, Kubectl, Jenkins, Python, AWS, Azure, CI/CD scripted Pipelines, terraform, Kibana, ElasticSearch. Prometheus, Distributed Tracing, SSL certificate management, nginx.

Skills Required for Monolith: Apache, WildFly, Redis, Jenkins, Python, AWS, Azure, CI/CD scripted Pipelines, terraform, Kibana, ElasticSearch. Prometheus, Distributed Tracing, SSL certificate management.

eGov Services implementation overview

eGov

eGov develops software for state government, based on specific requirement like citizen, tax etc., from each state and delivers software only for that specific requirement.

The client will give their AWS or azure or bare metal account, which is the environment, they want to deploy cluster. Dev & QA team has complete access to all modules. UAT & production environment managed for client will have only required modules.

Devops

Devops is a mindset or culture of developing & deploying software.

Other SDLC models focus only the development of software. But in Devops, it is just the first stage. The important phase is deployment. Devops guides us how development team & operations team can securely release the product with process automation.

Devops lifecycle: Plan->Code->Build->Test->Release->Deploy->Operate->Monitor

In regular monolithic applications, we create a single WAR file based on the code given by developer, which is manually deployed in tomcat server, by stopping the server, clearing cache & then restarting tomcat server.

In devops, we deploy maven artifact in docker container instead of directly in a tomcat like server.

In Devops, once the developer finishes code, instead of manual steps, we use Maven for writing all the steps required to build in the form of automation script, which will run all tests & automatically create an artifact (jar or war file). Command used for this purpose is 'Maven build'

CI / CD pipeline is created using Jenkins, which will automate the maven build process & deploy into the correct environment.

For deploying build in multiple environments, instead of changing values of the variables in the configuration file manually, CI/CD tools will automate them as well. Thereby, reducing work burden of developer as well as operations team & speedup release cycle.

After automating the process, wherever it is possible, then the remaining work will be done by Devops engineer.

Jenkin pipeline steps for a module written in Java - Billing service

1. In Jenkins, give the command -> '**Build now'**. Repository (having the updated source code), configured in GIT will be pulled.
2. Now workspace is updated with new code and we have to give the command ->'**Maven Build'**. It will run unit tests and generates JAR file.
3. **Pull** a base image from docker repository, copy your WAR or JAR file to the image and build as new docker container image using the command -> **'Docker Build'**
4. Give a tag id to the new customized docker image & **Push** to docker repository. Once docker image reaches docker repository, it means that build deployment is successful.

The above steps can be automated using shell script. It will get executed as a single step process. But pipeline scripting involves doing the steps separately, so that trouble shooting is easy. Freestyle scripting option is also available. But pipeline scripting is used popularly.

Microservices

In earlier days, for monolithic/ legacy application deployment, company will purchase a server and then when traffic increases, they will increase additional servers for load balancing. Auto scaling & load balancing was difficult with servers, when the traffic on the application increased.

Then people started using virtual machines (VM) concept based on AWS cloud computing. Load balancing, auto scaling option is also available in VM. Suppose we are creating VM for AWS, it cannot be used in Azure or Bare metal. So VM is suitable for monolithic applications, but not for microservices.

Nowadays we create docker container image instead of VM, which can be deployed anywhere. Each build history can be taken as a separate docker container image. Suppose the latest build deployment doesn't reflect in production, it is easy to revert to the earlier build version and deploy it for production. In monolithic application, only the latest copy of war file is only available in tomcat server. It's not easy to revert to earlier build versions.

In microservices, all are plugabble modules. We can use only the modules required, instead of loading all modules in a single war file, like in the monolithic application.

Microservices can be tested independently. Each microservices can be developed in different technology - one in java spring boot, one in php, one in nodejs etc. and finally run them in the same environment.

There is a possibility of integration issues, when we individually deploy microservices. So, we have to develop modules separately and keep them as dependent modules and prepare the chart of dependent modules in the early stage itself.

Suppose there is a USER module, it will be used by all other modules, to check whether that particular user exists and then provide services. We will develop USER as individual module and other modules will be dependent on it.

Also, in monolithic application, we have to install required software (JVM for Java) in server first, before you can execute any application. When we install more applications in server, it becomes overloaded and slows down performance.

Instead, in microservices, we will create docker container image for each application which can be deployed in any platform (physical or cloud).

Docker container image file is a repository of required files for running application. It is generated by using the command -> **'Docker Build'.** Instead of 'Docker Build', if we give the command -> '**Docker run',** it will create a container in local environment using the docker image. It will have separate ip address and port number. **When the image is in running condition, it is called as container.**

In GIT source code available for all modules, whereas in docker container image, only the required modules are available. In version control system like Git, all developers will be committing code to same branch, once the code is ready.

For using microservices in mobile devices, it involves APK upload. The docker container image will be deployed as APK file in APP store.

For especially public grievance (PGR), mobile apps are used. Whether it is mobile app or website, same microservices will be called, but the front end (UI) will be different.

Kerala govt, Andhra govt are still running in monolithic architecture.

Kubernetes

Kubernetes is a tool which help to package microservice container, with required resources and do the deployment in required environment. Kubernetes system is made up of clusters. Each cluster has a cluster master and multiple nodes. Each node runs pods.

Earlier, cluster is deployed using KOPS (Kubernetes Operations) which involves, creating cluster master in client env/work env of Kubernetes. Now we are using managed services called AKS (Azure Kubernetes Service) from azure as well as EKS (Elastic Kubernetes Service) from AWS.In AKS/EKS, We need not create cluster master. It will be managed by cloud environment. Only nodes are managed by devop team. Whenever you want to scale up, we need to create node and add to cluster master.

Cluster master - is responsible for deciding what runs on all of the cluster's nodes. This can include scheduling workloads, managing the workloads' lifecycle, scaling, and upgrades. The master also manages network and storage resources for those workloads. It intelligently handles distributing work to the individual nodes for you. If any nodes are added or removed, the cluster will shift around work as necessary.

Node - worker machines that run containerized applications and other workloads and may be either a virtual or a physical machine, depending on the cluster. Each node has the services necessary to run pod.

Pod - A Pod is the basic execution unit of a Kubernetes application. It may have one container or **multiple containers**. A Pod always runs on a Node. If there are more than one pod, it will get executed automatically in round-robin system.

Suppose a container crashes due to some reason, it will not recover automatically. Then it will not have a separate ip address and DNS discovery. Pod can restart a container.

In Jenkins, if you give the command -> **deploy to dev,** it will automatically take server identification, user name & password given in configuration file, & **kubernetes** will deploy the docker image in that dev environment

Database deployment

PostgresSQL Database will be created in RDS - Relational database service in AWS cloud. There is automatic master/slave, automatic backup, easy upgrade & downgrade, continuous monitoring of metrics regarding CPU utilization etc.

For a new service deployed, we have to provide login access for users. For example, existing user will be given permission to access new service.

In this case, 2 containers will be run. First container is called DB container or init container. Only when init container runs successfully, application container will be deployed. SQL scripts which gives permission will be stored in init container and will be executed first before application runs.

Source code:

For eGov project, source code is maintained in 2 folders in Github -

1. microservice modules (open source, available for both devops team & public)

2. infraops (credentials, data required to deploy, available only for devops team)

After coding by developer, each functionality is delivered as a microservice. Each microservice is a docker container & deployed in kubernetes cluster.

Internal workflow of how source code is built & deployed:

In any Spring based application, there will be **COLLECTIONS folder**

COLLECTIONS folder -> src folder, Docker file, pom.xml, start.sh

The dev team will write all code inside src folder

pom.xml will used for updating the libraries to be included.

Egov has their own **Nexus repository** to store libraries, which will be referred by pom.xml

Dockerfile & start.sh is used for building docker image for deployment. Inside dockerfile, we can see references to image file, jar file and start.sh file. Start.sh file will run java command to start the application

In monolithic applications, when we run a java application locally, we create Maven build. The maven build will create an artifact like Jar file. Jar file will be deployed in tomcat server.

In this project, we run the maven build. It will create a maven artifact (Jar or War file), we copy the maven artifact into java based docker image & push the image into docker repository. That image will be deployed in kubernetes environment.

This is how 90% of java modules are build and deployed

Similar to collections folder, there is one more folder in github repository -> **WEB folder**

WEB folder -> src folder, Dockerfile, pom.xml, start.sh

WEB folder will have resources required for UI / Frontend. When we are using NodeJS as frontend technology, all npm libraries are stored in a customized image in a private repository. Node & npm libraries are preloaded in the base docker image & we use that base image to build our customized docker image.

When we run the npm build (source code is created using NodeJS), it will create an artifact with static content, in the form of zip file, which will be hosted in Ngnix, which will serve static content. Ngnix is an API gateway manager.

Either it is AKS or EKS, there is a gateway which will interact with external users. The gateway is ELB (Elastic Load Balancer) for EKS. From ELB it will go ngnix gateway. Ngnix gateway will have all API and front end location. Redirection to front end or back end location is managed by Ngnix.

ELB-->Ngnix-->Zuul-->API's

For example, if we are using Punjab egovernance website, and accessing any functionality API--it will come through ELB and then Ngnix gateway. It will redirect to collections docker container. After Ngnix, it goes to API gateway manager called Zuul. It manages all API authentication & authorization.

If there is an API, there is a user login and according to their user access permission & restrictions, they are allowed to do any activity. A database is maintained for user privilege for each module. This is taken care by Zuul (all API related transactions)

If it is front end, it will not go to Zuul, it will directly go to containers. Only for backend API it will go to Zuul and then to respective API.

There is one more folder in github repository -> **INFRAOPS folder** (access by only devops team)

In Kubernetes deployment, for each service, we have yaml file, which is a deployment configuration file. Here we will override the variable values during deployment. For each env, we can pass values to variables with respect to env, which means it is dynamic and not static. The values differ from env to env. We do configuration & customization together. Configuration changes from Ngnix based deployment, Java based deployment.

Any module apart from core module is maintained separately. Backbone is **namespace**, which is also maintained separately.

Elastic search is maintained separately with log data & persistent data. **Kibana** will be used as dashboard to access the data, maintained separately.

In the asset index of yaml, we can see how many replicas, what is image name, variables that have to overridden for environment release / deployment (Variable values are different in dev env & production env). pom.xml will have default values which has to be overridden.

Whenever there is deployment, two images will be sent -> front end image, database image

In the database image, add all queries, which must be executed on database, before any starting any service.

For example, if billing service users have to be given access during deployment, privileges for that billing service will be added to database image, and will be executed first, before billing service starts.

For each service, there will be different environmental configuration.

Config maps - variable for particular service. Common variable like DB URL, index will be stored.

Secrets - store data in encrypted way like password.

Volumes - if there is persistent data (data accessed frequently), attached to any service, we can create stateful set, which deploys the pod again in the same machine, to access the attached volume.

Kubernetes has the ability to change host automatically, whenever new deployment of the same service occurs. If you have 5 nodes and 2 replicas. Whenever you deploy same service, it will get deleted and create as new docker image, not necessarily in the same host. It can be deployed to different host. In this way, it is possible that persistent data will get lost, when you have a storage volume, attached to a particular node. For example, for Kafka, it will understand that, whenever Kafka deployment is happening, it has to attach volume to the particular pod and for next deployment, it will not miss data.