QuickBase DevOps Task

by Petar Denev, 2021

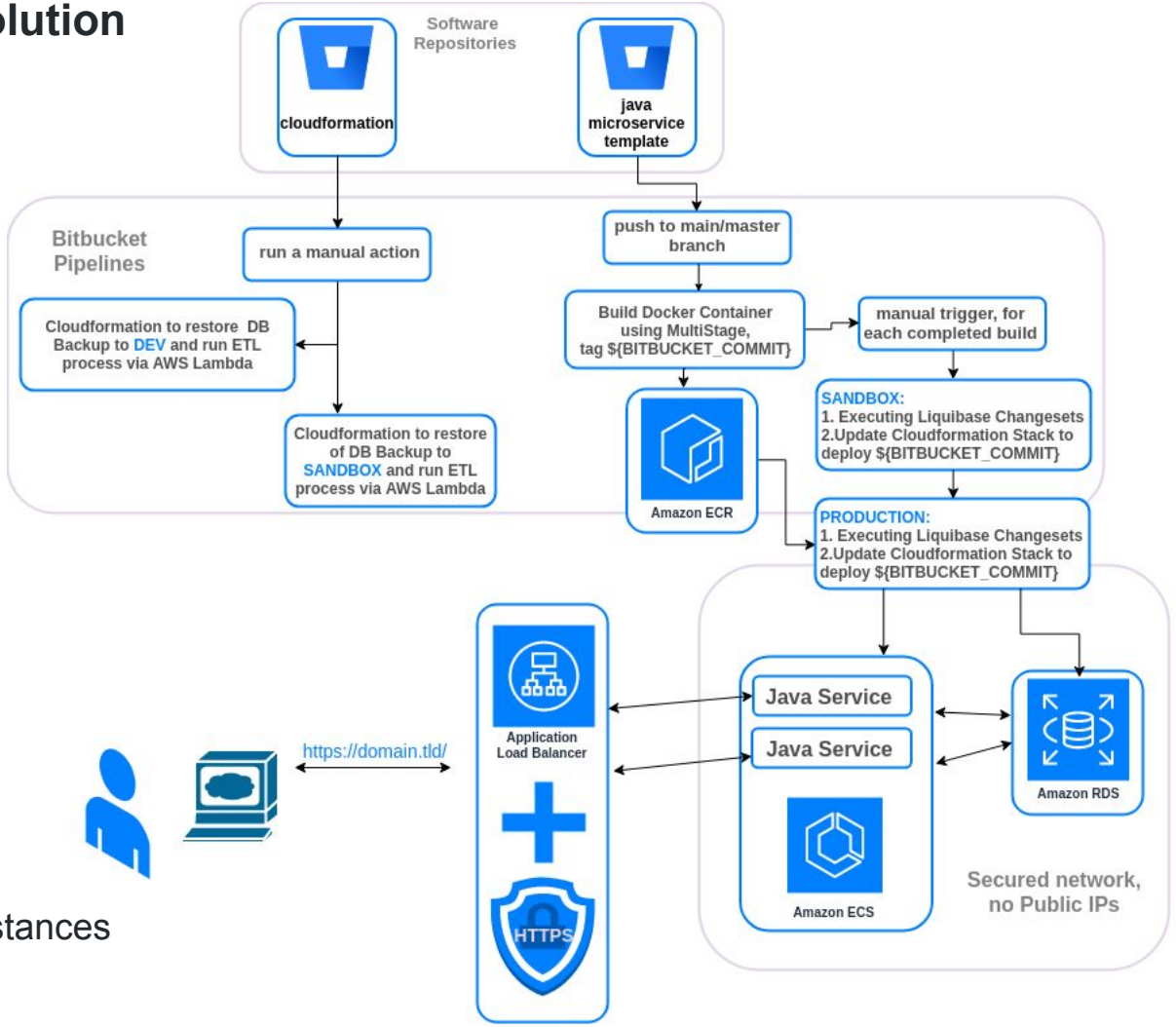
Contents:

- Brief description of the solution
- What type of steps you would perform in order to verify the deployment is successful?
- Plan and a task break-down how you would implement monitoring of this deployed app
- What kind of security policies and scans would you recommend to put into place?
- What other improvements would you make to the CI/CD process if you had more time?
- Are there any other good dev-ops tools and practices that you recommend?

Brief description of the solution

1. Solution Architecture involves the use of the following technologies:

- AWS Cloudformation
- Docker MultiStage
- Bitbucket Pipelines
- AWS ECR
- AWS ECS
- AWS ALB
- AWS RDS
- Java 11
- Maven3
- AWS Lambda
- Liquibase
- AWS Cloudwatch
- AWS SSM to access the instances



Brief description of the solution

2. Implementation Technique valid for **java-microservice-template/bitbucket-pipelines.yml**

- The results of using the java-microserive-template repository would allow the devops engineer to have one implementation valid for all Java Services part of the application stack
- The deployment logic is done in a way that it requires on first deploying the particular commit (represented as Docker Container) to Sandbox Environment and then, only if the previous step has completed successfully, the engineer is able to deploy to Production Environment, this is a fail safe mechanism, preventing non-working solution to get to Production
- Again, the deployment won't happen, if the Liquibase Changes don't apply
- The use of the docker cache does allow for faster build process
- Deployment happens with executing the **aws/update_stack.sh ${repository-name}-sandbox sandbox aws/${repository-name}.yaml ${git_commit} ${repository-name}**
- Application Logs are exported to AWS Cloudwatch group named **/${repository-name}/docker/${env_name}** where ${env_name} can be sandbox or production
- Cloudformation template and update_stack.sh are located in the **java-microservice-template/aws** directory
- Liquibase automation part of the deployment logic are located in the **java-microservice-template/db** directory, named **Dockerfile** and **update_db.sh**
- In the process of deployment, if the container does not start, its automatically stopped(the old version is not being shut down, until the new version is verified working), this is done with adding a HealthCheck endpoint verification from the Application Load Balancer, if the container does not start for 30 seconds - its reverted

What type of steps you would perform in order to verify the deployment is successful?

- Have the Load Balancer to check the health endpoint of the Service:

```
LBTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    Port: 8080
    Protocol: HTTP
    VpcId:
      Fn::ImportValue: !Sub "${InfrastructureStackName}:VpcId"
    TargetGroupAttributes:
      - Key: deregistration_delay.timeout_seconds
        Value: 30
    HealthCheckIntervalSeconds: 60
    HealthCheckPath: /health-check
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 15
    HealthyThresholdCount: 3
    UnhealthyThresholdCount: 10
    Matcher:
      HttpCode: 200
LBListenerRule:
  Type: AWS::ElasticLoadBalancingV2::ListenerRule
  Properties:
    Actions:
      - Type: forward
        TargetGroupArn: !Ref LBTargetGroup
    Conditions:
      - Field: host-header
        HostHeaderConfig:
          Values:
          - !Ref DNSRecordV4
    ListenerArn:
      Fn::ImportValue: !Sub "${InfrastructureStackName}:LoadBalancerListener"
    Priority: 1
```

Plan and a task break-down how you would implement monitoring of this deployed app?

- Have the deployment verified by the Load Balancer with the HealthCheck, as shown on the previous slide
- Have the application logs exported to AWS Cloudwatch, as shown on the screenshot below, located in **aws/java-microservice-template.yaml**:

```
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      RetentionInDays: !Ref LogsRetention
      LogGroupName: !Ref LogGroupName
```

- Have the application logs exported to /dev/stdout in the **server/src/main/resources/application.yaml** file, spring deployment profile:

```
logging:
  level:
    org.springframework.web: INFO
  file: /dev/stdout
```

What kind of security policies and scans would you recommend to put into place?

- Except the actual infrastructure implementation described already in the cloudformatio/infrastructure directory, in particular:
  cloudformation/infrastructure/00-infrastructure-core.yaml
  cloudformation/infrastructure/01-acm-route53-auto-approver-lambda.yaml
  cloudformation/infrastructure/02-infrastructure-application.yaml
  cloudformation/infrastructure/cf-helper-ELBv2-rule-priority.yml
  cloudformation/infrastructure/task-execution-assume-role.json

- Use cloud native tools, in case with AWS https://aws.amazon.com/blogs/containers/amazon-ecr-native-container-image-scanning/
- Have the public endpoint to be attached to a Load Balancer, not to the container itself
- Have WAF implementation https://aws.amazon.com/waf/features/
- If needed, review the https://aws.amazon.com/compliance/services-in-scope/ and implement https://aws.amazon.com/compliance/csa/
- Have rotation of the AWS credentials for the account that is used to manage the infrastructure
- Have an ability to use temporary security credentials for accessing the Cloud Service https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html

What other improvements would you make to the CI/CD process if you had more time?

- DB Anonymization Process, allowing the developers to work with anonymized database
- Implement Selenium-Grid https://www.selenium.dev/documentation/en/grid/ for executing QA Automations
- Implement Performance Testing of the Staging(Sandbox) Environment with JMeter https://jmeter.apache.org/ and/or Gatling https://gatling.io/
- Implement JProfiler https://www.ej-technologies.com/products/jprofiler/whatsnew12.html to help the developers debug their application better
- Implement AWS Xray for Production Environment https://aws.amazon.com/xray/

Are there any other good dev-ops tools and practices that you recommend?

- Try to follow The Twelve Factor Application Principles in building the DevOps Software Solution https://12factor.net/ for example: https://hub.packtpub.com/how-to-build-12-factor-design-microservices-on-docker-part-1/ https://hub.packtpub.com/how-to-build-12-factor-design-microservices-on-docker-part-2/
- Try to write the devops software solution implementation, so that most of the code is being re-used, best solution would be to have a template repository for each different programming framework used, like the current implementation hosted in the java-microservice-template directory
- Make sure you use immutable infrastructure, otherwise you might run into a lot of problems
- Implement SonarQube https://www.sonarqube.org/ to catch bugs and vulnerabilities in your app, with thousands of automated Static Code Analysis rules
- Implement PagerDuty https://www.pagerduty.com/platform/event-intelligence-and-automation/ for Incident Management, On-Call Management
- Implement Datadog https://www.datadoghq.com/ or Grafana https://grafana.com/ with Prometheus https://prometheus.io/ (depending on the business use and project size)
- Provide the developers with the ability to fully re-create the environment on which they work, so they can access the resources and be 100% sure that the problems they're experiencing are not related with the setup of the infrastructure