Containers for Customer Success

Migration and Managing micro services
Monolith Apps

- Huge stack to manage
- Complex Code base
- Integration testing is a Pain
- Slow Development Process
- Provisioning Infra-services is an Issue
- Build – 2 – deploy cycles are high
- Horizontal Scaling is a pain
- Change management is an Pain
- HIGH DOWNTIME - During deployments and troubleshooting
Why Micro services

- Controlled Code Base
- Language Agnostic
- Data decentralization
- Horizontal Scaling
- API based communication
- Deployments Pipelines are faster
- Detailed Logging
- Reliability, Scalability, Manageability
Containers

- Isolation
- Run anywhere
- Highly Scalable
- Automated Rollback's
- Easy to Provision
- Easy to Build and Deploy
- Highly Consistent
- Very less time to boot
- Monitored and metered resource utilization
Best Practices with Containers:

- Always use smaller images, migration & movement is easier
- Always use Docker file in your continuous integration.
- Version control your docker files.
- Migrate docker file with artifacts

Image Request
- Application version
- OS Version
- Patches
- User and directory structure

Image Creation Process:
- Dockerfile
- Bake the Image
- Launch and verify the container

Vulnerability Scan:
- Vulnerability scan
- Identified VA fixes will be applied
- Certified Image created.

Upload to Registry
- Linux tools installed by default
- Images posted to central registry

Certified OS Images and Sizes
- RHEL 7 - 200Mb
- Alpine OS - 5Mb
Continuous Deployment with Containers

1. Pull an Certified Image
   - Pull Image from Equinix Docker registry
   - Create / Request generic docker file
   - Always version control docker file

2. Build a Container
   - Add deployment steps docker file
   - Executing docker file will create a app container
   - Docker file with built artifact is a best practice

3. Run App Container
   - Execute docker file in any environment
   - Release image can be created from running App container
   - Scan and store the release image if needed

Jenkins
Continuous Integration

- Maintain a code repository for each micro service.
- Automate your build using CI Tools.
- Make your build self-testing.
- Have Integration and Unit Testing.
- Every commit (to the baseline) should be built.
- Keep your builds fast.
- Have Docker file / Image creation process for the micro service.
- Integrate it with Build.
Continuous Deployment of Microservices

- Developer
  - Forked repository
  - Equinix Repo Developer branch
    - Merge to Dev branch
    - Webhook
  - GitLab
    - Code review
    - Approves MR
  - Reviewer
    - Approves MR

- Jenkins
  - Create Merge Request
    - Webhook
  - Build Job
    - Run Static code analysis
    - Publishes to Sonar
    - Code compilation, code coverage
    - Merge conflicts and create the artifacts
  - On Success
    - Publishes to Sonar
  - Merge to Stable branch
  - Continuous Integration

- SonarQube
  - Analyzes code quality
  - Unit test case results
  - Notify Developer and reviewer

- QA Deployment
  - Functional test cases
  - Regression, security test cases

- Prod Deployment
  - All test cases

- Continuous Delivery
Build and Deploy – Pipeline – Microservices

Build and Container Image creation
Deploy in DEV & Test
Deploy in INT & Test
Deploy in UAT & Test
Deploy in PRODUCTION

Build and Deploy – Pipeline – Monolith Apps
Manageability

1. Every Micro service is a container
2. Resource Scaled On-Demand
3. Health Check on Service level
4. Port Isolation
5. Horizontally Scalable
6. Hosted based on Affinity
7. Secured using container network
Scaling - Micro services

1. Auto-Detect every micro service
2. Manage with sticky session @ Load Balancer level.
3. Name based container registration
4. Service check
5. Automatic container update & configuration.
6. LIVE Configuration updated
7. Environment segregation
Micro services as Catalogues

1. Every Micro service is a container
2. Every Micro service is a catalogue
3. Access controlled Deployment
4. 0 Downtime upgrade
5. Horizontally Scalable
6. Load balanced Initially
7. Isolated on the Host
Benefits

**Improvement**
1. Developer Agility
2. Patching and Tracking
3. Code and Service Quality
4. Faster Deployments
5. Matured Code-base
6. Resiliency
7. Scalability
8. Isolate and fix

**Efficiency**
1. Zero Downtime Deployments
2. Business visibility
3. Scale Globally
4. Roll over upgrades
5. API maturity
6. MULTI-CLOUD deployment
7. Self-Serviceability

**Visibility**
1. Single Pane of Glass
2. Projects Quality
3. Status of Feature
4. Resource and people utilization
5. Service Availability
6. DevOps Dashboard
7. Continuous QA
8. DEV-OPS Platform

**Security**
1. Environment Abstraction
2. Secured Across
3. Controls at every layer
4. Monitored for anomalies
5. Data Isolation
6. Control & Operate
Monitoring and Manageability

INFRASTRUCTURE

CI/CD

TEST

Develop

All Developers Favorite Language and Technology

Selenium
NGINX
Apache
JUnit
JMeter
JUnit
CentOS
redhat
docker
sonarqube
git
Maven
gradle
Jenkins
git
Maven
Ansible
Rancher
Vagrant
Rundeck
kubernetes
Prometheus
dynatrace
CONFIO
gomez
logstash
ScienceLogic
Seagull
Hyperic
packetbeat
AppDynamics