B4M36SWA: Software Architectures

Deployment of traditional Java EE applications in the cloud

Jiri Pechanec
QA Engineer

May 2nd, 2017

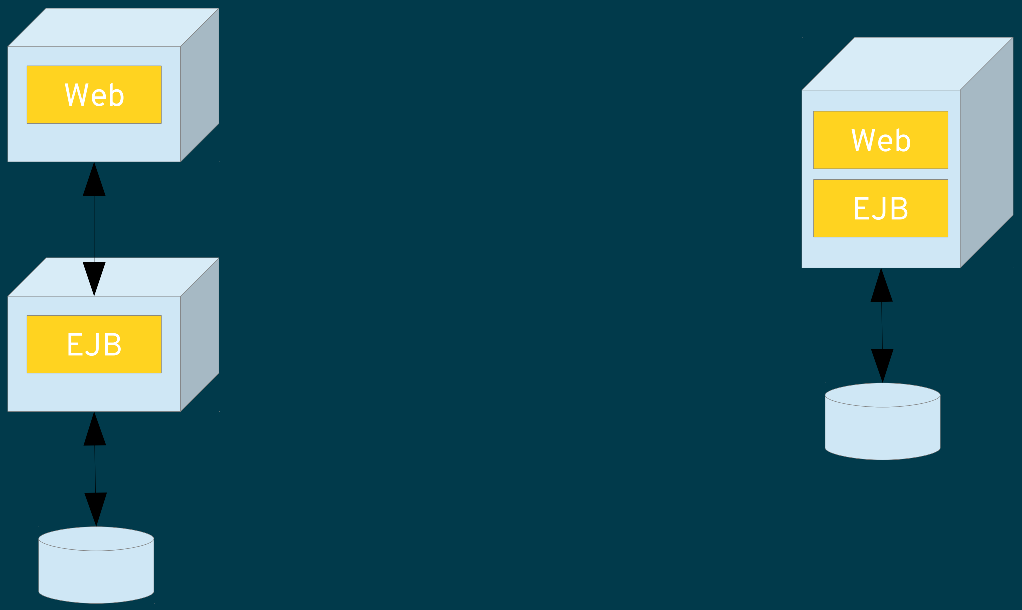
AGENDA

- Java EE Container
  - Standard deployment
  - Clustered deployment
- Keeping the state
- Deployment in cloud
  - VMs
  - Containers
  - Scaling
    - Manual
    - Automatic

Java EE Application

- Java application running in Java EE container
- Application
  - Stateful 3-tier
    - Data Tier – standalone SQL Database
    - Logic Tier – EJBs, CDI beans
    - Presentation Tier – JSF, Servlets, JS frameworks

Java EE Application Deployment



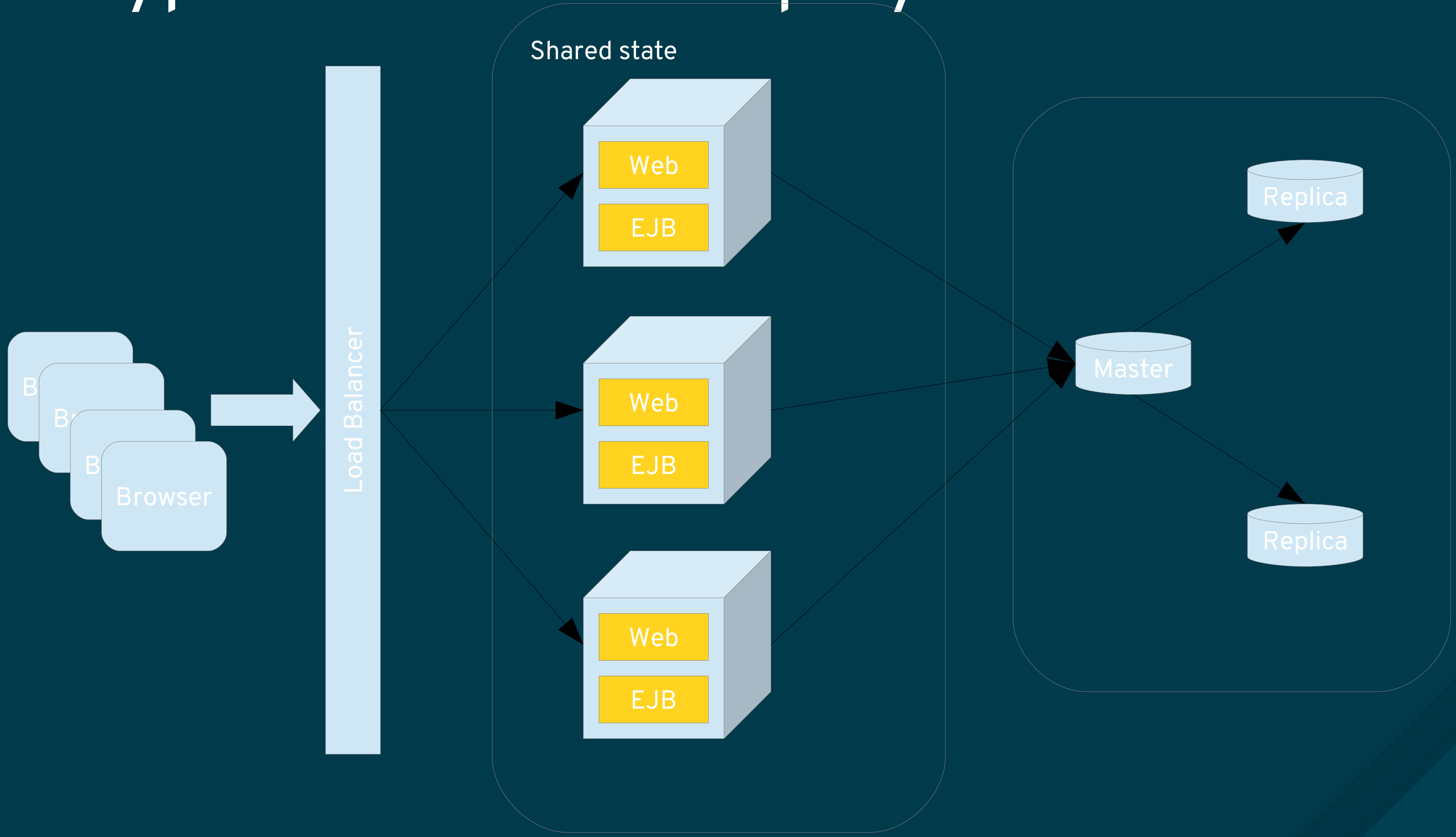
Java EE Container

- Java runtime that provides services for Java EE applications
- Supports multi-application deployment
  - Not too used – mostly as application modules
- An application archive is placed into a deployment directory
  - EAR
  - WAR
  - JAR

Java EE Container

- Clustering
- Usually a static number of replicas on bare-metal or virtual machines
- Discovery problem
  - Static list of IPs/hostnames
  - Service registry
  - UDP Multicast

Typical clustered deployment



Typical clustered deployment

- Database should be replicated
  - Single point of failure
- Load balancer routes web client requests to an instance of application server
  - random
  - round-robin
  - load
- App server instances contains state
  - Must be replicated between instances
  - Sticky session optimization

What is state

- Typically associated with a web session
- Represents an interaction between application and the user
  - Shopping cart
  - User session
- An application server itself can keep an internal state
  - HTTP Session
  - Caches for JPA
  - Stateful Session Bean instances

Application deployment process

- Developer commits
- A final integration build
- Testing in a stage environment
- Passing the application artifacts to operations
- Outage
    - Cluster is stopped
    - A new version of the application is copied into the deployment directory
    - Start the cluster

Moving to cloud

Dynamically provided VMs

- First stage of moving to cloud
- Needs automatic cluster establishment and discovery
  - WildFly/JBoss EAP – UDP multicast
  - Load balancer must also support dynamic configuration
    - Cluster instances advertises their presence to load balancer
- Cloud provider provisions a new VM
- A VM is configured
- App server is started and automatically joins the cluster

VM Configuration

- Manually
- Automatically
  - Ansible, Puppet, …
- Changes
  - Install the app server
  - Configure the server
    - DataSources, JDBC Driver, JCA connectors
- Deploy the application
- Time consuming and error prone process

Pre built-VM images

- Part of the application release process
- When a new version is released a pre-configured VM is built
  - Amazon AMI building
  - Oz for OpenStack
  - packer – multiplatform
- Adding a new image to cluster means starting the customized image

Container as a Service

- Cluster instances are running in Docker containers
- Similar to pre-built VM, just Docker image is built
- Allows much higher application density than VM-based solution

Platform as a Service

- The platform takes care of the whole release process
- The platform is capable of executing Maven builds
  - A location to an SCM is provided
  - A build can be triggered on-demand or automatically
- Automatic application re-deployment when a new container image is available

Java EE Application in OpenShift

Main concerns

- Automated build
- No outage deployment
- Dynamic clustering
- On-demand scaling
- Keeping the state

Automated build

- Source-2-Image (s2i) images are used to build to application
  - WildFly, JBoss EAP
  - Tomcat
- S2I executes Maven build and bakes the application into the image with Java EE container
- BuildConfig points to application sources
  - Manual trigger
  - Webhook trigger
  - Rebuild when new s2i image is available (CVEs)
- DeploymentConfig points to the image built by OpenShift
- Automatic application re-deployment when a new release is available

No outage deployment

- DeploymentConfig points to the image built by OpenShift
- Automatic application re-deployment when a new release is available
  - recreate – outage
- Rolling update
  - new nodes are started and are joining the cluster
  - old instances are removed from the cluster
  - active sessions are migrated between new and old nodes
  - Requires forward and backward compatibility!!!

Scaling

- Manual
    - oc dc <> --replicas=
- Automatic
    - Horizontal pod autoscaler object
    - Web UI – deployment → Actions → Add autoscaler
        - Minimum and maximum of pods
        - CPU threshold
    - Prerequisites
        - OpenShift Metrics deployed in cluster
        - Resource limits for CPU set

Keeping the state

- Database
  - no-brainer
  - requires persistent volume for database datafiles
  - MySQL, PostreSQL, MongoDB
- Replicating the state between WildFly/JBoss EAP instances
  - Application must be <distributable/>
  - Internally uses Infinispan project – a shared cache

Database deployment

- Deploy database exposed as a service <name>-<databasetype>
- Deploy a JBoss EAP with env vars
  - DB_SERVICE_PREFIX_MAPPING: <prefix>= <name>-<databasetype>, …
  - <prefix>_JNDI
  - <prefix>_USERNAME
  - <prefix>_PASSWORD
  - <prefix>_DATABASE
  - TX_DATABASE_PREFIX_MAPPING
    - transaction manager log storage
- Multiple databases supported

Transaction Manager concerns

- Transaction Manager controls 2PC transactions
- Requires a transaction log to bookkeep transactions in progress
- Problem with scaling in case of using file system
  - Nodes come and go
- File-based approach with database bookkeeping

KUBE_PING

- Service discovery
  - KUBE_PING protocol for JGroups networking stack
- Requires *view* privilege for service account or project
  - oc policy add-role-to-user view system:serviceaccount:$(oc project -q):<account name|default> -n $(oc project -q)
- Env vars for pod must be set
  - *OPENSHIFT_KUBE_PING_NAMESPACE* – name of the project
  - *OPENSHIFT_KUBE_PING_LABELS* – labels to identify pods to be merged in a cluster
- Container port 8888 must be exposed

KUBE_PING

- Service registry based discovery
- OpenShift provides a list of pods based on label filter
- Each pod exposes a simple HTTP server on port 8888 for cluster forming communication

Supplementary services

- Remote cache
  - A standalone running instance of Inifinispan (JBoss Data Grid)
    - REST
    - HotRod
    - memcached
  - Application stores its state here instead of a database
  - Hibernate OGM
- Messaging
  - Java Message Service

Java Message Service

- Asynchronous communication
- Decouples source and target
- Two models
  - queues – point-to-point
  - topics – publish/subscribe

HornetQ

- A component of JBoss EAP
- Automatically clustered
- Requires env vars
    - MQ_CLUSTER_PASSWORD
    - MQ_QUEUES
    - MQ_TOPICS
- Can deliver messages between pods in the cluster

JBoss A-MQ

- A standalone message broker, multiprotocol
  - Openwire, AMQP, MQTT, STOMP
- Based on Apache ActiveMQ
- Static scaling only
- JBoss EAP can automatically replace an internal HornetQ with a link to a standalone JBoss A-MQ

JBoss A-MQ deployment

- Deploy a JBoss A-MQ instance, exposed as a service <name>-amq
- Deploy a JBoss EAP with env vars
  - MQ_SERVICE_PREFIX_MAPPING: <prefix>= <name>-<databasetype>, …
  - <prefix>_JNDI
  - <prefix>_USERNAME
  - <prefix>_PASSWORD
  - <prefix>_PROTOCOL
  - <prefix>_QUEUES
  - <prefix>_TOPICS
- Multiple brokers supported

Summary

Summary

- Elasticity of cloud bot simplifies and complicates application deployment
- Key differences between traditional and cloud deployment
  - Deployment pipeline
  - Elastic scaling
  - Application state management

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHatNews

youtube.com/user/RedHatVideos