Microservices + DevOps + Oracle Cloud = A Bright Future

Sai Janakiram Penumuru
Chief Technologist, Oracle ACE Director
Sai Janakiram Penumuru
- Chief Technologist – Apps Transformations - DXC Technology
- Co-Founder, Vice President - All India Oracle Users Group (AIOUG)
- Oracle ACE Director, Oracle Developer Champion
- Oracle VM SIG Leader www.oraclevmSIG.org
- Blog: www.oadba.com; www.oracle12c.info
- Contacts – ps.janakiram@gmail.com; twitter - @saiPenumuru
Agenda
For 45 Minutes

– Microservices
– DevOps
– Oracle Cloud
What is Microservices

• The first one, Microservices is a software architecture style in which complex applications are composed of small, independent processes communicating with each other using language-agnostic APIs.

• The second one describes Microservices Architecture as a style of architecture that promotes business alignment by developing applications as set of small independent and self-contained services that directly caters to an atomic business activity.

Martin Fowler: Microservices
Microservices are Analogous to **Unix Utilities**

Same Concept, Different Decade

* Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*

- Unix Executable: Does one thing and does it well
- Runs independent of other commands
- Produces text-based response

```
curl -v -H "Accept: application/json" -H "Content-type: application/json" -X POST
-d '{"productId":645887,"quantity":"1"}'
"http://localhost:8840/rest/ShoppingCart/"
```

- Microservice: Does one thing and does it well
- Runs independent of other microservices
- Produces text-based response to clients

Doug McIlroy
Inventor of the Unix Pipe
Monolithic versus Microservices

A monolithic application puts all its functionality into a single module

... and scales by replicating the module on multiple servers

A microservices architecture puts each element of business functionality into a separate service

... and scales by distributing these services, in parallel, across servers, replicating as needed.
Monolith vs Microservices

- Single, Monolithic App
- Must Deploy Entire App
- One Database for Entire App
- Organized around Technology Layers
- One Technology Stack for Entire App

- Many, smaller minimal function Microservices
- Can deploy Each Microservices independently
- Each Microservices often has its own Data store
- Organized around Business capabilities
- Choice of Technology for each Microservices
Fundamentally

Do you want...

Traditional App Development

Easier Deployment/Ops
- One big block of code, sometimes broken into semi-porous modules
- Complexity handled inside the big block of code
- Each big block is hard to develop but easy to deploy

Microservices

Easier Development
- Many small blocks of code, each developed and deployed independently
- Complexity encapsulated in each microservice
- Each microservice is easy to develop but hard to deploy
Microservices Apps Are Developed/Deployed Independently
Conway's Law in Action

Any piece of software reflects the organizational structure that produce it
Successful Teams Structure their teams around Products

Build Small vertical teams
What is the right level
Let’s make breakfast - objective of the use of Microservices is to increase re-use.

The right level of decomposition is Business and Organization dependent. What that entity executes as an atomic business function.
Characteristics of Microservices
Organized around business capabilities

Traditional Approach

Microservices Approach

Presentation Layer

Application Layer

Database Layer

Siloed functional Teams

Cross-functional Teams

Services are organized around business capabilities, including user interface, persistent storage and external collaboration.
Traditional Transaction
Let’s go to the pub

A session remains open during your presence at the pub.
Transaction less coordination between services
The Starbucks model

If something goes wrong compensating operations are used to address the issue. Failure does not result in loss of state as Microservices are stateless. Failure is imminent, graceful recovery is paramount.

By making the link between Microservices asynchronous, having one service going down does not affect the operation of the other. Synchronous calls are considered harmful. If one service no longer respond, the other keeps hanging for a response and becomes unavailable.
Characteristics of Microservices
Design for Failure

Traditional Approach

Microservices Approach

500 Internal Server Error

1. Service down, recognize the failure, spin up a new parallel instance or do graceful degradation.
2. If the service is slow, timeout, either act (spawn a new parallel instance to compensate for extra workload) or report to the user the unavailability and potentially allow him to retry.
3. Service responds with unexpected content, is your service capable of recognizing and handling this? Microservices must have a stable and published contract.
4. When-ever possible, use caching mechanisms so the service can still respond if the back-end is down.

Unfortunately the XYZ service is currently down, we apologize for the inconvenience. However our remaining services are operating normally.

Graceful recovery and process resilience are mandatory in the architecture and design of Microservices based solutions.
What is required for Microservices?

Rapid Provisioning
Monitoring
Rapid Application Deployment
DevOps Culture
DevOps
Familiar?
DevOps seeks to solve this
Dev and Ops Constantly Argue

*Code is written... it’s your problem now*
Shift in priorities is demanding DevOps

One product team!
- Shared objectives, Shared customer-oriented goals, Shared accountability
Building a continuous application development supply chain
Create “BizDevOps”

Per “What is DevOps” by dev2ops.org
It worked fine in dev... it’s ops’ problem now.
Today’s application delivery is full of obstacles and challenges

- Siloed Teams, Lack of end to end visibility
- Error prone manual hand-offs and processes
- Manual and error prone app deployments
- InfoSec & compliance engaged late driving vulnerabilities & re-work
- “patch in production” leads to snowflake systems
- High # defects
- Poor confidence in test data fosters “release aversion” driving more WIP
- Isolated build and integration processes
- Long waiting time to setup and test environments
- Lack of effective customer insight and high latency drives “kitchen sink” requirements
- Rapidly increasing WIP

One way flow:

1. Business Demand
2. Planning
3. App Development
4. App Testing
5. Release decision
6. App release
7. Deployed App

Business Demand → Planning
Planning → App Development
App Development → App Testing
App Testing → Release decision
Release decision → App release
App release → Deployed App
Deployed App → Internal Customers
Deployed App → Internal Customers
Deployed App → External Customers

The Market

Revenue

Costs

Poor user experience

High defects

Manual Testing increases latency or drives limited test coverage

Internal Customers

External Customers

$ Revenue

$ Costs
YOU FOOL!
It’s not about the tools, it’s about the CULTURE!
### Change in Culture

#### Traditional IT Development Life Cycle

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Long and rigid planning process related to current and historical needs</td>
<td>Step-by-step delivery in a waterfall or production line approach</td>
<td>Departmentalized roles working in silos and throwing jobs over the wall</td>
<td>Changes are problematic as they need re-defining, re-scheduling and re-working, leading to delays and increased costs</td>
<td>Slow and costly manual deployments and implementations</td>
<td>Working in silos leads to a narrow view aligned with limited or initial requirements</td>
</tr>
</tbody>
</table>

#### Agile/DevOps Development Life Cycle

<table>
<thead>
<tr>
<th>Adaptive</th>
<th>Goal Focused</th>
<th>Teamwork</th>
<th>Responsive to Change</th>
<th>Automated</th>
<th>Business Focused</th>
</tr>
</thead>
<tbody>
<tr>
<td>React and responds to changing needs constantly</td>
<td>A business oriented approach and attitude which sees &quot;the bigger picture&quot;</td>
<td>Collaborative and integrated methods to empower teams across the business and the client</td>
<td>Agile methods quickly identify areas for change or improvements</td>
<td>Automated deployment tools and virtualized environments to accelerate releases</td>
<td>Working in teams leads to a holistic view in line with today's business needs</td>
</tr>
</tbody>
</table>

#### Comparison

<table>
<thead>
<tr>
<th>Define</th>
<th>Build</th>
<th>Test</th>
<th>Release</th>
<th>Plan</th>
<th>Define</th>
<th>Build</th>
<th>Test</th>
<th>Release</th>
<th>Plan</th>
<th>Define</th>
<th>Build</th>
<th>Test</th>
<th>Release</th>
<th>Plan</th>
<th>Define</th>
<th>Build</th>
<th>Test</th>
<th>Release</th>
<th>Plan</th>
<th>Define</th>
<th>Build</th>
<th>Test</th>
<th>Release</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Do DevOps?

– enabling faster feature time to market
– increased customer satisfaction & market share
– employee productivity and happiness
– organizations to win in the marketplace

In contrast, organizations that require weeks or months to deploy software are at a significant disadvantage in the marketplace.
PaaS/IaaS Now Allows Resources to be Easily Provisioned
DevOps Principles
Cultural movement enabled by technology

Dev
Paid to add new features

Ops
Paid to keep system stable, fast and available

DevOps
New goal:
Add new features and keep the system stable, fast and available
Characteristics of DevOps Movement
Principles have been around for decades

Movement began in startup community. Use of open source seen as integral but not technically necessary.
DevOps = Culture + Technology Movement
Culture is what’s behind DevOps; Technology is the enabler
DevOps Tenet #1: Culture

- Dev respect for ops
- Ops respect for dev
- Don’t stereotype
- Don’t just say “no”

Respect

- Don’t hide things
- Ops should be in dev discussions
- Dev should be in ops discussions
- Shared runbooks/escalation plans
- Ops should give devs access to systems

Discuss

- "Done" Means Released
- Dev’s responsibility does not end when it’s in production
- “Throwing it over the wall” is dead

Avoid Blaming

- No fingerpointing!
Build Respect

• Developers should respect Operations
• Operations should respect Developers
• Don’t just say “no”
• Don’t stereotype!
Discuss

- Don’t hide things! Open communication on both sides
- Operations should be in Developer discussions
- Developers should be in Operations discussions
- Build shared runbooks/escalation plans
- Operations should give Developers direct access to systems
Avoid Blaming

• No fingerpointing!
• Development should have enough
  Operations culpability to share the
  blame in an outage
• Operations should have enough
  Development culpability to share the
  blame in an outage
**Actively Build Trust**

Trust is the #1 ingredient to a successful DevOps culture

Dev + Ops + Social Activity Outside Work + Time = Trust
DevOps Tenet #2: Technology

- **Infra as Code**
  - Use config mgmt to build environments
  - Scripts checked in and managed as src

- **Shared Version Control**
  - Single system for code and build artifacts
  - Every time someone commits a change, it triggers a build and automated build verification tests
    - Ship trunk
    - Enable features through flags

- **Don’t Fix Anything**
  - If something breaks, re-deploy. Don’t fix
  - Fix environment setup scripts

- **One Step Build/Deploy**
  - One button build/deploy (manual)
  - Scheduled builds/deploys
  - If verification fails, stop and alert
Infrastructure as Code
Manage it as you would any other source code
Shared Version Control
Surprisingly not well adopted

- Single system for code and build artifacts
- Every time someone commits a change, consider triggering a build + automatic verification tests
- Always ship trunk!
- Enable features through flags
One Step Build/Deploy
Set it and forget it

- Manual one button build/deploy
- Scheduled builds - every day, every week, etc
- Builds triggered by code checkins
- If post-build validation fails, report it
Automated Testing using Robot

1. Integrate Robot With Maven

```xml
<plugin>
  <groupId>org.robotframework</groupId>
  <artifactId>robotframework-maven-plugin</artifactId>
  <version>1.2</version>
  <executions>
    <execution>
      <id>robot</id>
      <phase>integration-test</phase>
      <goals>
        <goal>run</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

2. Write a Simple Selenium-based Test

```java
*** Settings ***
Library
testSetup
testTearDown
```

```java
*** Test Cases ***
Basic Test
Open Browser
Page Should Contain
Page Should Contain Textfield
Page Should Contain Textfield
Page Should Contain Button
Input Text
Input Text
Click Button
Wait Until Page Loaded
Page Should Contain
Close Browser
```

3. Run Test

`mvn verify`

4. View Results

```text
[INFO] --- robotframework-maven-plugin:1.2:run (robot) @ my-basic-
```

```text
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
```

```text
Acceptance
```

```text
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
```
## Core Tools required

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
<th>Sample Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Code Management</td>
<td>Central Repository for all code</td>
<td>Source Code Mgmt</td>
</tr>
<tr>
<td></td>
<td>All code (application, ops, qa) checked-in</td>
<td>Oracle Developer Cloud</td>
</tr>
<tr>
<td></td>
<td>Version control, labelling and release management</td>
<td></td>
</tr>
<tr>
<td>Continuous Build &amp; Integration</td>
<td>Centralized system used to schedule and control jobs</td>
<td>Hudson</td>
</tr>
<tr>
<td></td>
<td>• Build</td>
<td>Jenkins</td>
</tr>
<tr>
<td></td>
<td>• Integration and Deployment</td>
<td>Oracle Developer Cloud</td>
</tr>
<tr>
<td></td>
<td>• Test jobs</td>
<td></td>
</tr>
<tr>
<td>Infrastructure as Code</td>
<td>Frameworks to enable the creation of automation to</td>
<td>Chef</td>
</tr>
<tr>
<td></td>
<td>• Provision VMs (OS Images, CPU, Mem…)</td>
<td>Puppet</td>
</tr>
<tr>
<td></td>
<td>• Storage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Ability to provision software, patches, dependencies dynamically</td>
<td></td>
</tr>
<tr>
<td>Cloud Infrastructure</td>
<td>Cloud provides for dynamic provisioning of Resources</td>
<td>Oracle Cloud</td>
</tr>
<tr>
<td></td>
<td>• VMs</td>
<td>Oracle Cloud Machine</td>
</tr>
<tr>
<td></td>
<td>• Storage</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PaaS Services (DBCS)</td>
<td>JUnit</td>
</tr>
<tr>
<td>Test Automation</td>
<td>Test Automation Frameworks</td>
<td>Selenium</td>
</tr>
</tbody>
</table>
Where to use DevOps
Analyze your portfolio
Oracle Cloud
Oracle can help you lead change in your organization.

But real change begins with you:

**DevOps Tenet #1: Culture**
- Respect
- Discuss
- Avoid Blaming
- “Done” Means Released

**DevOps Tenet #2: Technology**
- Shared Version Control
- Infrastructure as Code
- One Step Build/Deploy
- Don’t Fix Anything

How Oracle Can Help with DevOps:

- Simplify Installation
- Shrink Runtime
- Support Lifecycle
- Simplify Provisioning

Your responsibility, but Oracle can help.
Oracle Products support DevOps

DevOps Tenet #1: Culture
- Respect
- Discuss
- Avoid Blaming
- “Done” Means Released

DevOps Tenet #2: Technology
- Shared Version Control
- Infrastructure as Code
- One Step Build/Deploy
- Don’t Fix Anything
Oracle Developer Cloud Service – What’s In It

- Version Management – Git
- Build Automation
  - Ant, Maven, Gradle, npm, Grunt, Bower, Gulp, Command line
- Continuous Integration - Hudson
- Issues Tracking
- Code review
- Deployment automation
- Agile process management
- wiki
- Activity stream
Development Experience
Oracle Developer Cloud Service

- Effortless Project Management
- Teamwork Through Integrated Tools
- What You Need, Before You Need It
- From Just an Idea, to Product Release
Project Management

- Team members
- Activity stream
- Usage tracking
- Repositories
- Custom attributes
Requirements/Issue Tracking

• Create Requirements/Bugs/ERs
• Assign to team members and sprints
• Custom attributes
Agile Process Management

- Create dashboard
- Manage issues backlog
- Manage development sprints
- View team/tasks status
- Reports
Source Code Management

- Git repositories
- Branch, tag, merge
- Web interface
- View changes online
- Accessible from any Git client
- External repositories integration (for example GitHub)
- Snippets – for reusable code
Code Reviews

- Request code review
- Invite team members
- Comment on Code
- Accept / Reject / Iterate Reviews
- Merge Code
- Merge Conflict Resolution
Project Builds

• Maven
• Ant
• Gradle
• Node.JS – npm, grunt, bower, gulp
• Dashboard
• Logs and Audit
Deployment Automation

• Create deployment configurations
• Start/Stop a deployment
• Redeploy/Un-deploy applications
• In the cloud or on premise deployment
Continuous Integration

• Hudson
• Automate
  - Triggers
  - Schedule
• Dashboard
Wikis

- Share information
- Attachment support
- Wiki markup of choice
Merger of disciplines
Iterative Planning, Development and Release
Thank you

Sai Penumuru

Contacts – ps.janakiram@gmail.com ; twitter - @sai_penumuru