An Introduction to Microservices Architecture

BOV 1079

Rob Knight
rob.knight@suse.com
@rssfed23

Stephen Mogg
stephen.mogg@suse.com
@Stephen_mogg
Agenda

• Where we’re coming from

• The 12 factors – a mini deep dive

• How the public cloud can help
Where we’re coming from
In Today’s Era of Digital Transformation
Every company is a technology company

Technology has become a driver of competitive advantage across the business

No longer a mere supporting actor
Accelerate Delivery of Modern Applications
To compete and win in the digital era

**STABILITY**

- Increase responsiveness
- Speed innovation
  - Reduce cycle times, moving toward CI/CD (increase release frequency)
  - Deliver cloud native applications that are highly and dynamically scalable, resilient, and rapidly evolvable

**AGILITY**
Application Delivery Transformation Journey

**Containerize**
- Containerize & Orchestrate
- Accelerate application development and delivery
- Build and deliver new cloud native applications

**Modernize**
- Microservices & DevOps
- Modernize legacy applications

**Repeatable**
- Best practices, patterns, processes
- Discover & capture best practices, patterns, processes

**Standardize**
- Consistency across the enterprise
- Factory-scale application production
The Twelve Factors
The Twelve Factors – 1 - Codebase

One codebase tracked in revision control, many deploys

All your application code lives in one repository

Small codebases. Once a different end function is created a new repository is needed

Dependency management
The Twelve Factors – 2 - Dependencies

Explicitly declare and isolate dependencies

Never assume an app dependency is present on a host system

Use built in language package managers

Ship dependencies bundled into your app
The Twelve Factors – 3 - Config

Store configuration in the environment (not the code!)

Configuration is anything that may vary between different environments. Code is all the stuff that doesn’t

Store configuration as environment variables

Don’t bake credentials or dependent endpoint credentials in the code
The Twelve Factors – 4 – Backing Services

Treat backing services as attached resources

Backing service = anything the app consumes over the network

Local and third party resources should be treated the same

Loose coupling and soft failure
The Twelve Factors – 5 – Build, Release, Run

Strictly separate build, release and run phases

Build phase is for compiling code

Release phase is combining binariries with a specific target environment configuration

Separating them out allows developers to focus on compilation but the team running/operating the code have less complexities to deal with
The Twelve Factors – 6 - Processes

Execute the app as one or more stateless processes

Any persistent data should be stored outside of the app

The state of the whole system is defined by databases and shared storage, not individual running instances

Helps enable the scalability and highly available nature of “cloud native”
The Twelve Factors – 7 – Port Binding

Export services via port binding

The app should be completely self contained, not relying on an application or web server bundled with it.

Allows other apps to be the backing of other apps easily.

Web process is run entirely in userspace.
The Twelve Factors – 8 - Concurrency

Scale out via the process model

One process does one thing, and does it well

Factor less important these days, as rather than having one app with multiple process and scaling individual processes individually we recommend one process per service/app, and scale each component separately

One element still relevant, is avoiding daemonization and relying on the container orchestrator for stream and process management (example: k8s livenessProbe)
The Twelve Factors – 9 - Disposability

Maximize robustness with fast startup and graceful shutdown

Ready to serve requests as quick as possible

Processes should expect to be start and stopped at a moments notice (facilitates fast elastic scaling)

Be robust against sudden death
The Twelve Factors – 10 – Dev/Prod Parity

Keep development, staging and prod as similar as possible

Optimise for continuous deployment by closing the gaps

Make the time gap small: a developer may write code and have it deployed hours or even just minutes later.

Make the personnel gap small: developers who wrote code are closely involved in deploying it and watching its behaviour in production.

Make the tools gap small: keep development and production as similar as possible.

The same backing service (type) between environments.
The Twelve Factors – 11 – Logs

Treat logs as event streams

Don’t write logfiles or let the app worry about routing logs, log to stdout

Allows logs to be captured by the execution environment (such as k8s)
The Twelve Factors – 12 – Admin Processes

Run admin/management tasks as one-off processes

Admin processes should be run in an identical environment as production, by developers

Run against a release with same codebase and config as the final production environment the admin code will run in

Don’t run updates directly against a database, don’t run them from a local terminal window

Avoids issues using dev vs vendored versions of python, rails, etc that may have differences

Containers mitigate this somewhat and provide more flexibility. K8s has Jobs for this purpose
How the Public Cloud can help
Why Public Cloud for MicroServices?

On-demand (limitless*) resources

Experiment with low cost and risk

Speed

Service orientation

Support for Multiple Languages

* OK so there is a limit – you just have to work really hard to meet it
Public Cloud Services – more than just VMs
### Advantages of a Traditional Deployment vs. Advantages of using DBaaS

<table>
<thead>
<tr>
<th>Advantages of a Traditional Deployment</th>
<th>Advantages of using DBaaS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Familiar</td>
<td>Free up Staff</td>
</tr>
<tr>
<td>Existing Processes</td>
<td>Smaller organizations can take advantage (skills / $$$)</td>
</tr>
<tr>
<td></td>
<td>Scalability</td>
</tr>
</tbody>
</table>
Kubernetes as a Service
AKS, GKE and EKS

Are all completely awesome, but what if ..
• ...The CSP does not offer the version of Kubernetes you need for your application?
• ...You goal is to replicate an on-premises deployment of K8S?
• ...You need full control over the Kubernetes master server
• ...There is is a need to test out different / newer versions of Kubernetes?
Conclusion