Meetup

Application Modernization and Migration

Journey to the Cloud

OPENSHIFT

oc whoami

```
$ oc whoami
rmarting, jromanmartin
$ oc describe user rmarting jromanmartin
Name: Jose Roman Martin Gil
Created: 42 years ago
Labels: father, husband, friend, runner, curious, red hatter,
developer (in any order)
Annotations: Principal Middleware Architect @ Red Hat
Identities:
     mailto:   rmarting@redhat.com
     GitHub:   https://github.com/rmarting
     Twitter:  https://twitter.com/jromanmartin
     LinkedIn: https://www.linkedin.com/in/jromanmartin/
```

URBAN LEGENDS !?

DevOps !?

Devs

Ops

Application Designs !?

Monoliths

Microservices

What does APPLICATION MODERNIZATION & MIGRATION mean?

APPLICATION MODERNIZATION & MIGRATION

Focus on business workloads and solutions.

APPLICATION MODERNIZATION & MIGRATION

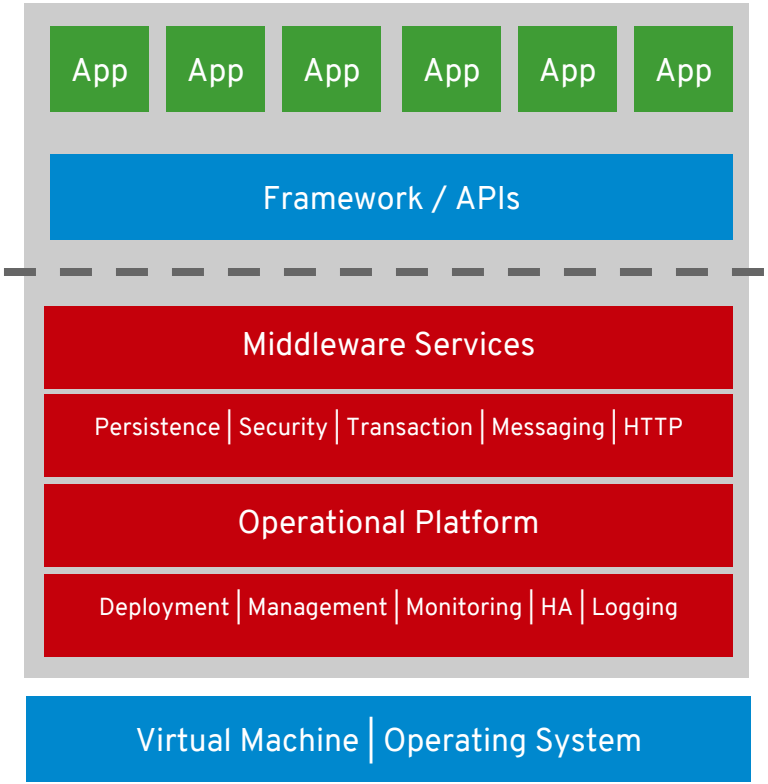
Digital transformation. Journey to the future.

APPLICATION MODERNIZATION & MIGRATION

Making old new again.

APPLICATION

Application Servers



Java™ COMPATIBLE ENTERPRISE EDITION

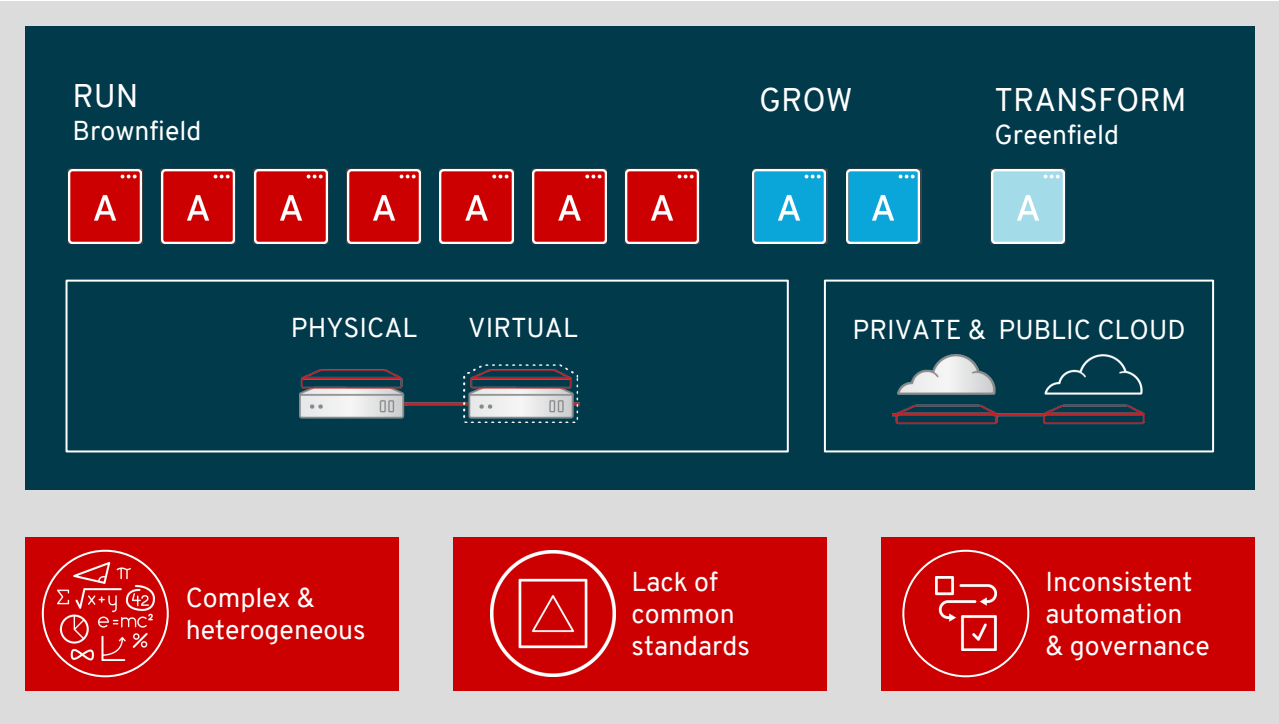
spring by Pivotal

JBoss® by Red Hat

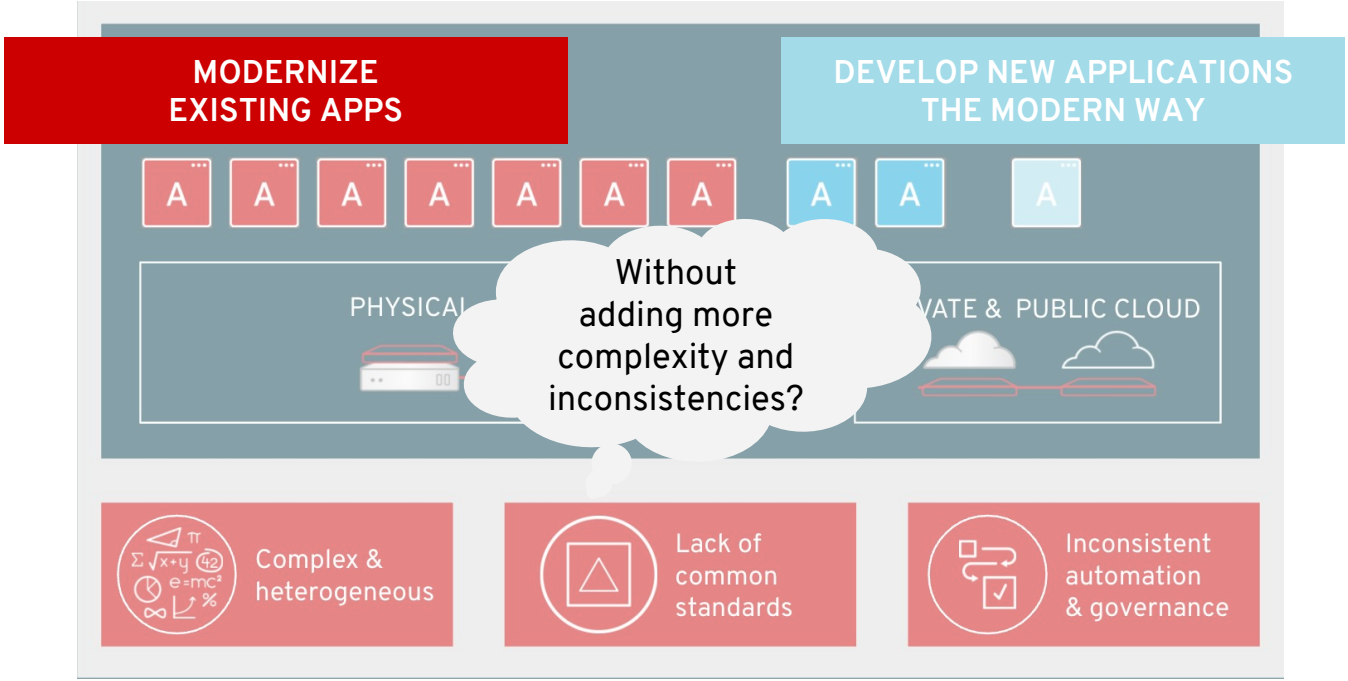
ORACLE®

WebSphere® software

Typical Landscape Today

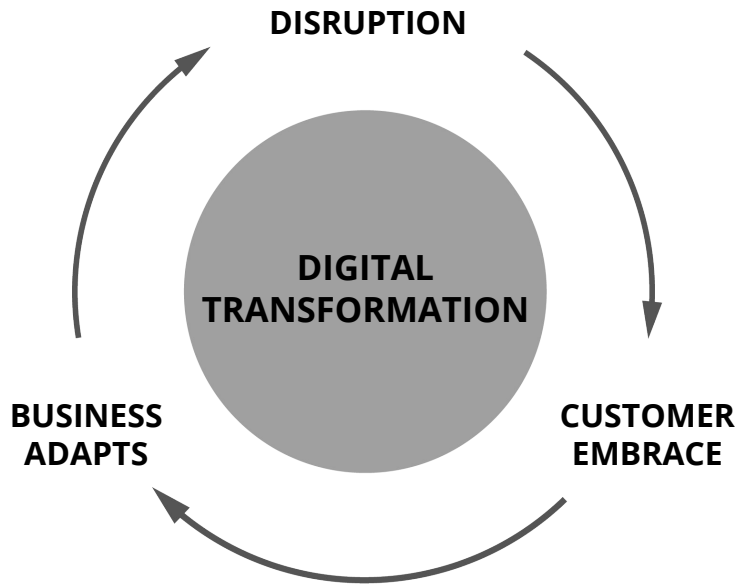


New problems to resolve



MODERNIZATION

Why Modernize?



Why Modernize?

- Every business is a technology business
- Code has no business value until it is deployed
- Scale and Speed challenges
- Competitive challenges
- Gain Business Agility
- Growth enablement

MIGRATION

Why Migrate?

There is not a “off button” for your Business Applications

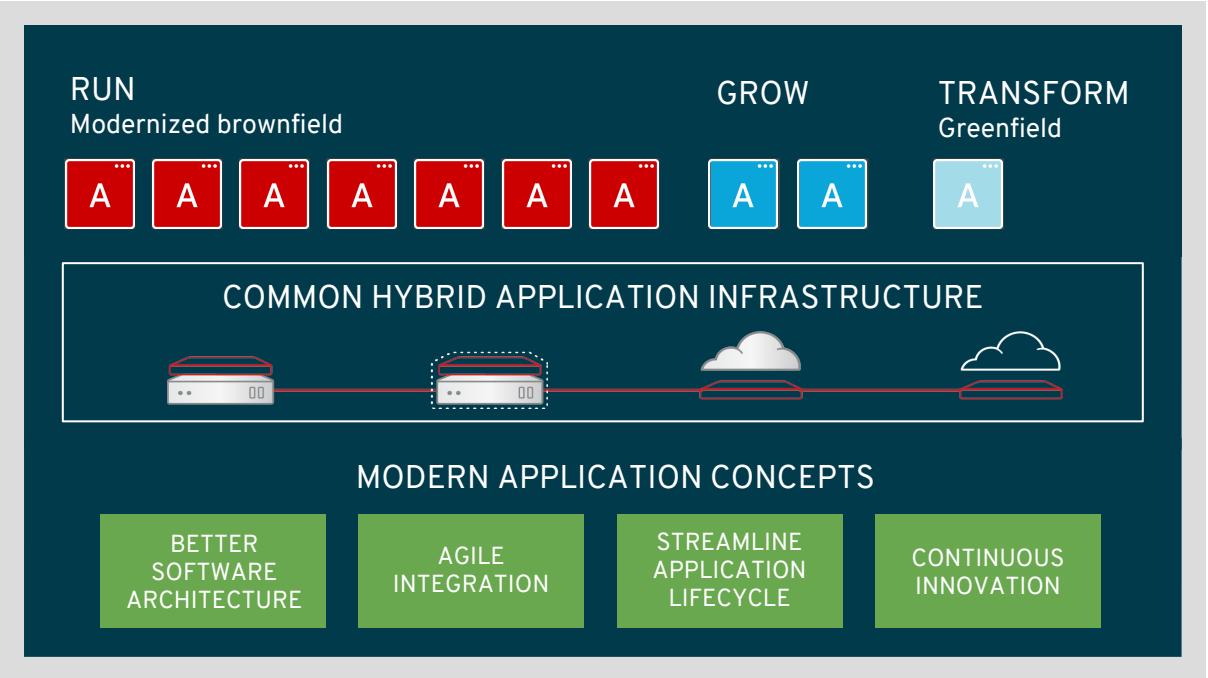
Why Migrate?

- Optimizing and streamlining existing application usage
- Unlocking more value from your IT investments
- Consolidate application instances
- Rehost or replatform the application to newer infrastructure
- Develop new application code to extend the life and utility of the legacy application
- Convert and update existing code into new development languages
- Restructure application code to support a more modular, loosely coupled services architecture
- Retire the existing packaged application and migrate to a new application

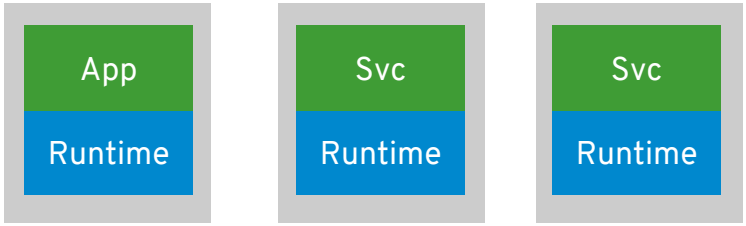
JOURNEY TO OPENSHIFT

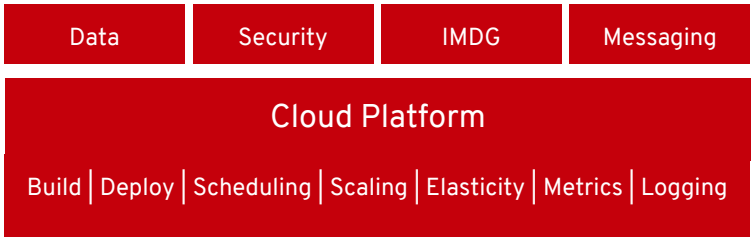
Where would like to be?

- One platform to support you today and tomorrow



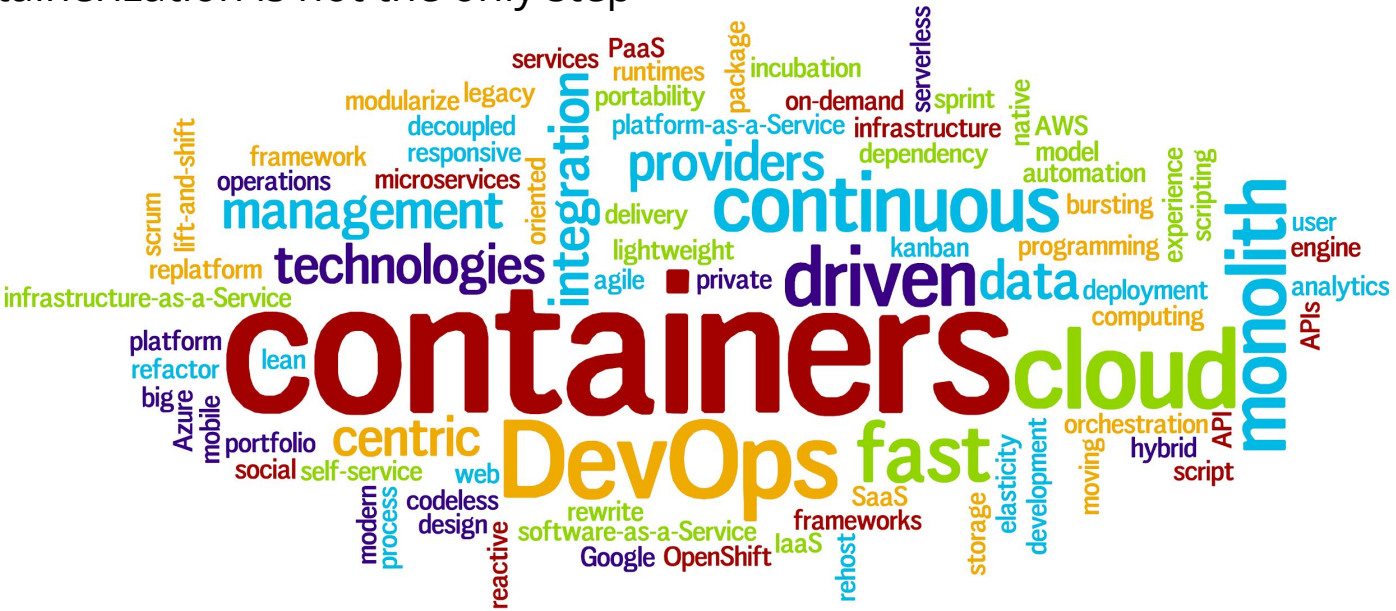
OpenShift is the new Application Server





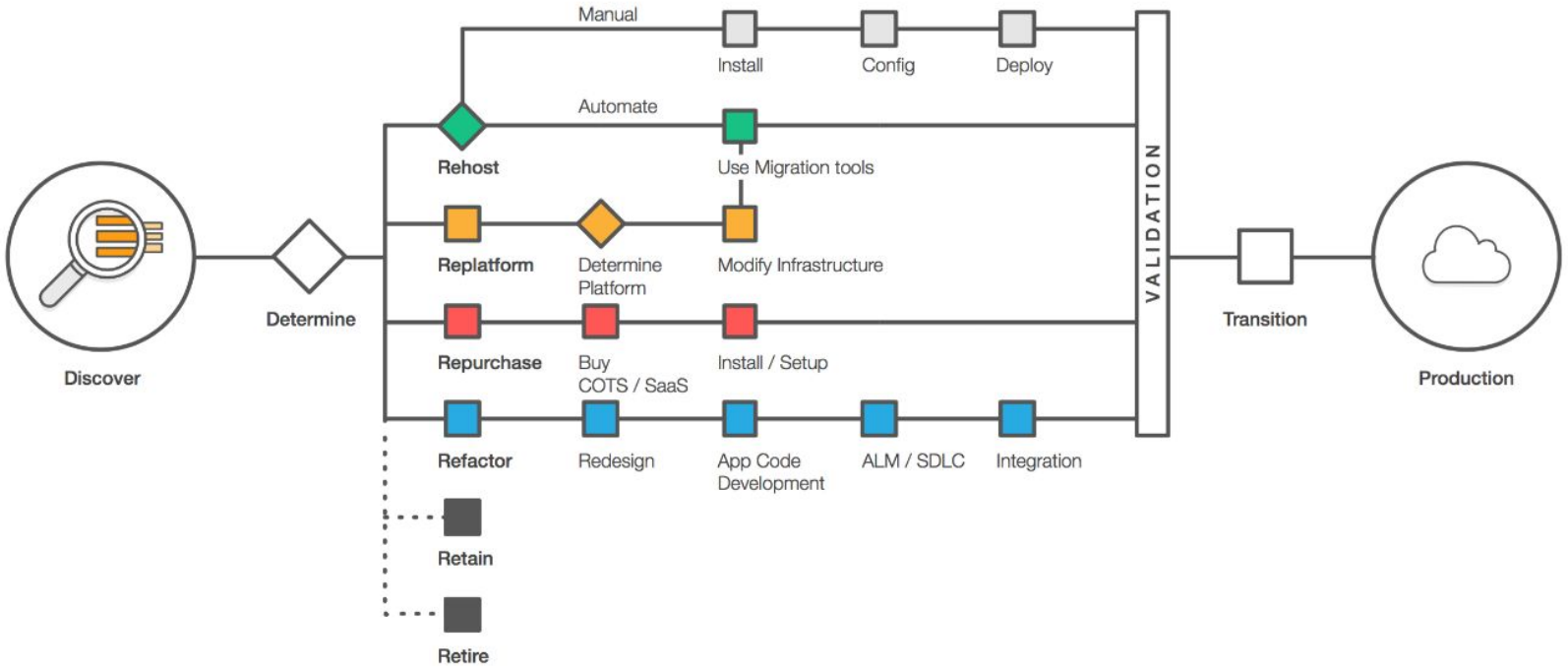
How do we run applications in the new world?

- Strategy and methodology
- Cloud Native, Cloud-Readiness
- Containerization is not the only step



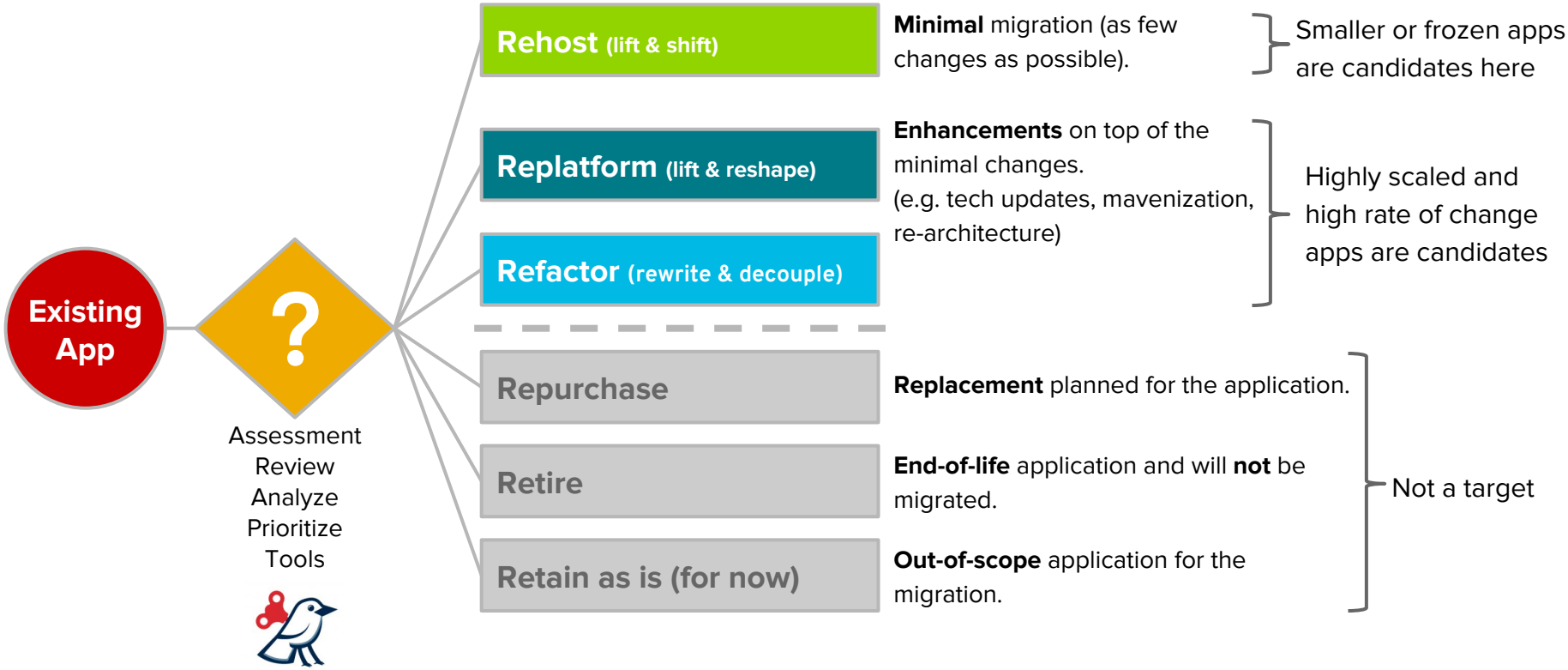
STRATEGIES

The 6 R's



Source: https://aws.amazon.com/cloud-migration/

Decision Tree



Patterns in Modernizing Applications

REHOST
(lift & shift)

- **Containerize** existing workloads
- Deploy them on a **PaaS**
- Keep external integrations and data on legacy
- Legacy applications have to be well written and suited

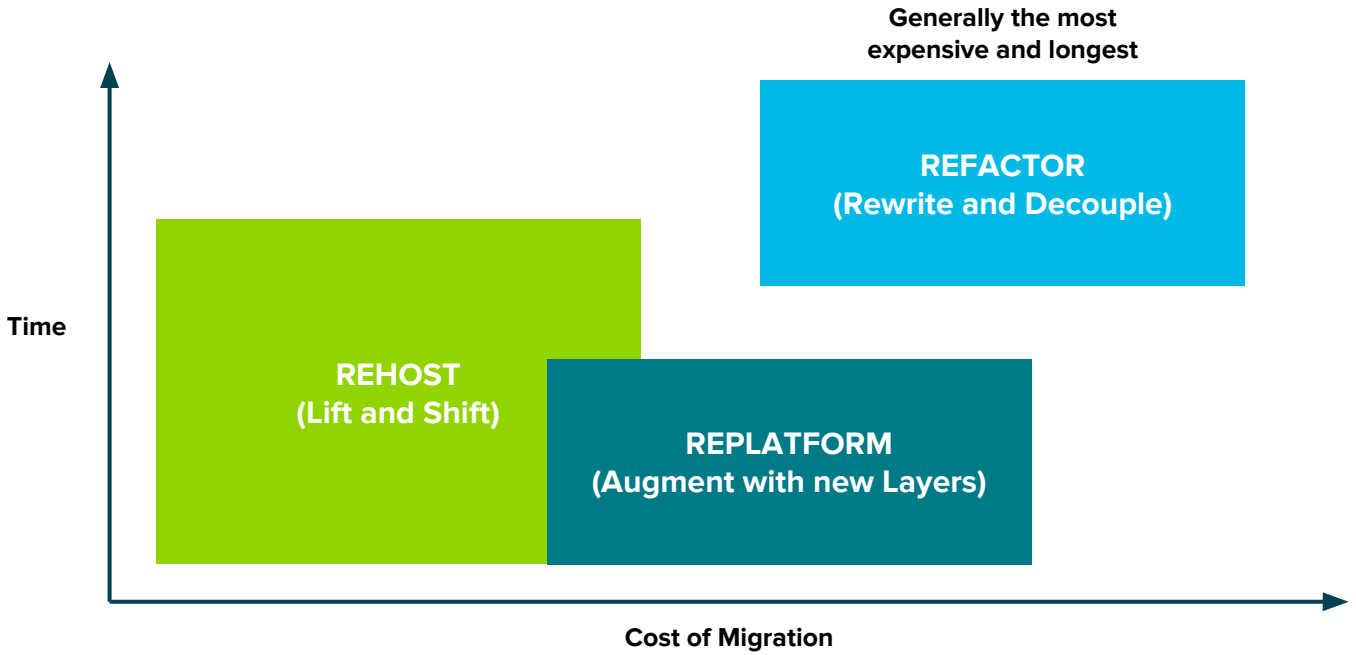
REPLATFORM
(augment)

- **Legacy remains intact**
- New layer - new capabilities
- Deploy on **PaaS**
- **New integration points** between legacy and new layers **(Need for Agile Integration)**

REFACTOR
(rewrite & decouple)

- **Legacy is totally replaced**
- New interfaces and data
- Use **PaaS** to run
- Some data and features can be re-wrapped, but mostly are retired.

Patterns in Modernizing Applications



Modern Application Concepts

Enhancing applications, platform and processes

BETTER SOFTWARE ARCHITECTURE	AGILE INTEGRATION	STREAMLINE APPLICATION LIFECYCLE	CONTINUOUS INNOVATION
Future-proof applications	Bridge old and new	Speed up your business	Foster an agile culture
Modularize "Fast moving monolith" Microservices Clean technical debt	Decouple, expose & integrate APIs, services & applications Need hybrid-cloud-enabled integration platform	Accelerate time from idea to production Continuous Integration & Delivery (CI/CD) Automation & self-service Container technology	Agile methodology DevOps principles Collaboration

CLOUD-READINESS

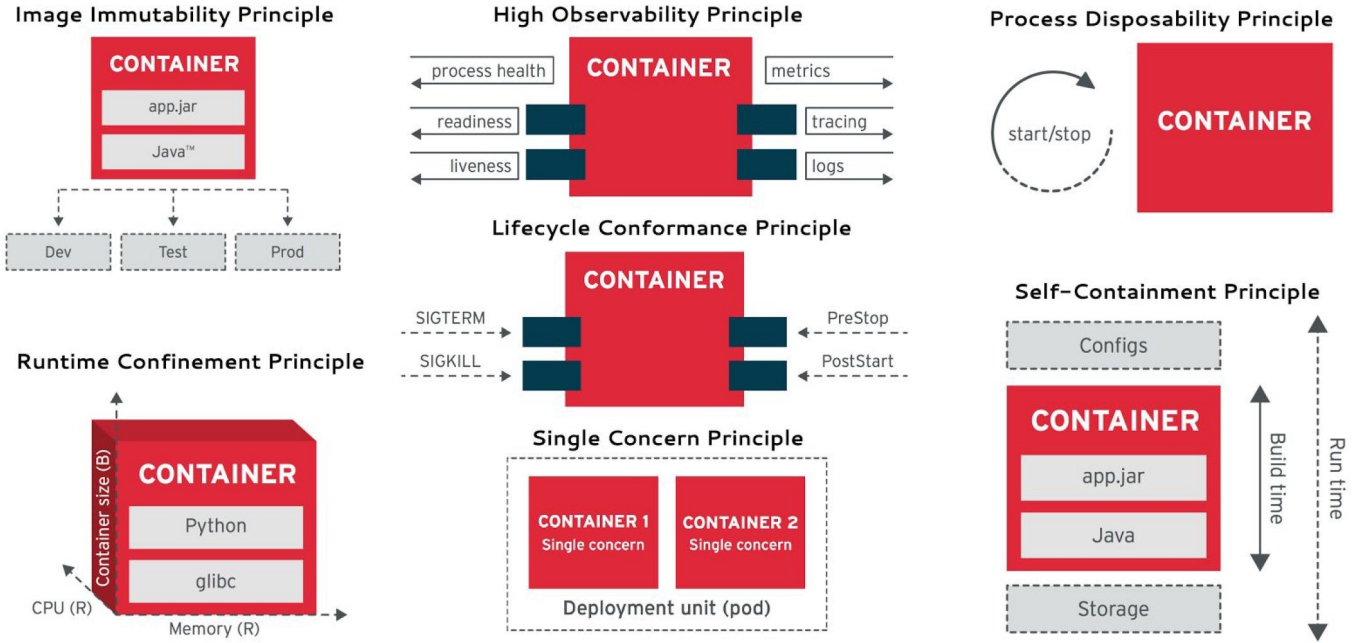
Cloud-Readiness

- “Cloud-Native”
  - Distinctive architectural characteristics
- “Cloud-Compatible”
  - Represents the minimum viable product
- “Cloud-Ready” / “Container-Ready” / “OpenShift-Ready”

- Not every application **should / can / must** be made “cloud-native”
  - Container-Ready is often enough
  - Primary focus on OpenShift adoption instead on high re-architecturing efforts

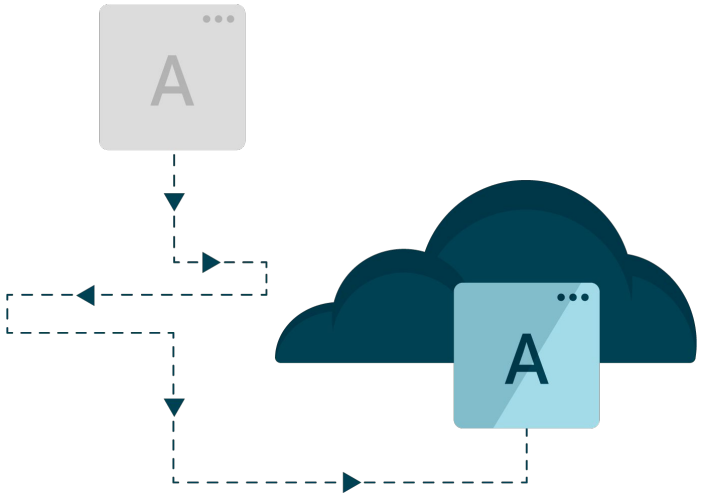
The OpenShift Effect

- [Principles of container-based application design]



Journey from Cloud-Compatible to Cloud-Native

- Start with Cloud-Compatible changes
  - Create support from external configurations
  - Remove IP bindings
  - Run on Linux
  - Ensure logs write to console/stdout
- Progress with subsequent iterations
- Actual end state should be dictated by the needs of the business



DEMO TIME

Demo Time - Show me the code

- Standard JEE Application
  - JPA + Hibernate to manage an external MySQL Database
  - JMS Queues to add more information to the application
  - MDB to load data from JMS Queues to store information into MySQL Database
  - Initial Servlets consuming extra startup time

- Main issues
  - Extra time to be ready after redeploy it
  - Monolithic architecture

https://github.com/rmarting/amm-demo

Demo Time - Migration

- Main Tasks
  - Package using Thorntail (aka Wildfly Swarm)
  - Deploy using Maven Fabric8 Plug-In

Demo Time - Thorntail

- An innovative approach to package and run Java EE applications
- *Just enough App Server*: package your app with required runtime dependencies (but nothing more)
- Based in Wildfly Application Server
- Packaged as an Uber Jar (self-contained, executable Java archive)

THORNTAIL

Demo Time - Maven Fabric8 PlugIn

- Brings your Java applications on to OpenShift
- Provides tight integration into Maven
- Focus on two tasks:
  - Building Docker Images
  - Creating OpenShift resource descriptors

Demo Time - Migration Results

- Health check of our application
  - Liveness & Readiness Probe
- Zero-downtime in deployment
  - Rolling up
- Scale up/down as needed
  - Autoscaller

Demo Time - Modernization

- Main Tasks
  - Deploy Messaging Services using ActiveMQ Artemis or enmasse.io
  - Deploy Integration Services with Apache Camel and Spring Boot
  - Deploy Cron Jobs
  - Remove old operations from monolithic application

Demo Time - Messaging Systems

- Messaging as a Service
- Messaging Protocols: AMQP, MQTT
- High Performance
- Scalable

Demo Time - Integration Systems

- Agile Integration - Cloud Native Integration
  - Integration Microservices
- Enterprise Integration Patterns
- Lightweight
- Data transformation

APACHE Camel

Demo Time - Modernization Results

- Services isolated and coordinated
  - Microservices?
- Fast monolithic application
  - Reduced the tasks done
- Resources Optimization
  - Each service uses the resources needed

Meetup

Questions?

OPENSHIFT

Meetup

Thank you!

OPENSHIFT