The Conference for Java
& Software Innovation
Oct 10 – 12, 2016 | London

The Asynchronous Uncoordinated Continuous Delivery of 35+ uServices

Clayton Wells | Ocado Technology

Talk Programme

- Ocado. Where we’ve come from and where we are going
- The Architecture needed to for Asynchronous Uncoordinated Continuous Delivery
- The Development Practices that enable Asynchronous Uncoordinated Continuous Delivery
- Q&A

Ocado

From a Online Grocery Company -> Technology Company -> Technology Provider

ocado -> ocado TECHNOLOGY

At Ocado

Ocado.com (before OSP)

- Deployments only every few weeks.
- Contained dozens of stories and bug fixes.
- Required days of testing.
- Always had bugs and required multiple hotfixes.
- Rolling out a feature meant deploying to subset of servers.

Ocado Smart Platform (OSP)

- Dozen of Deployments a Day.
- Contains Subtask of a Story.
- Testing Fully Automated.
- Small Bug Free Deployments.
- Fully in Control of the release programmatically.

The Architecture

- Stateless µServices
- Resilient
- Built in Fallback Strategies

Stateless µServices

- Statelessness
- Idempotent
- Single responsibility
- Keep databases local to uService

Resilient

- Not dependant on other systems to start up
- Don’t have processes built into the start cycle of the server
- Recover gracefully when dependant systems come online

Built in Fallback Strategies

- Build in how to do deal with unavailable services as a feature of the system
- Store static data from other services locally (ie configuration data)
- Hystrix, a nice tool for helping with fallback strategies

Architecture Recap

Software Development Practices

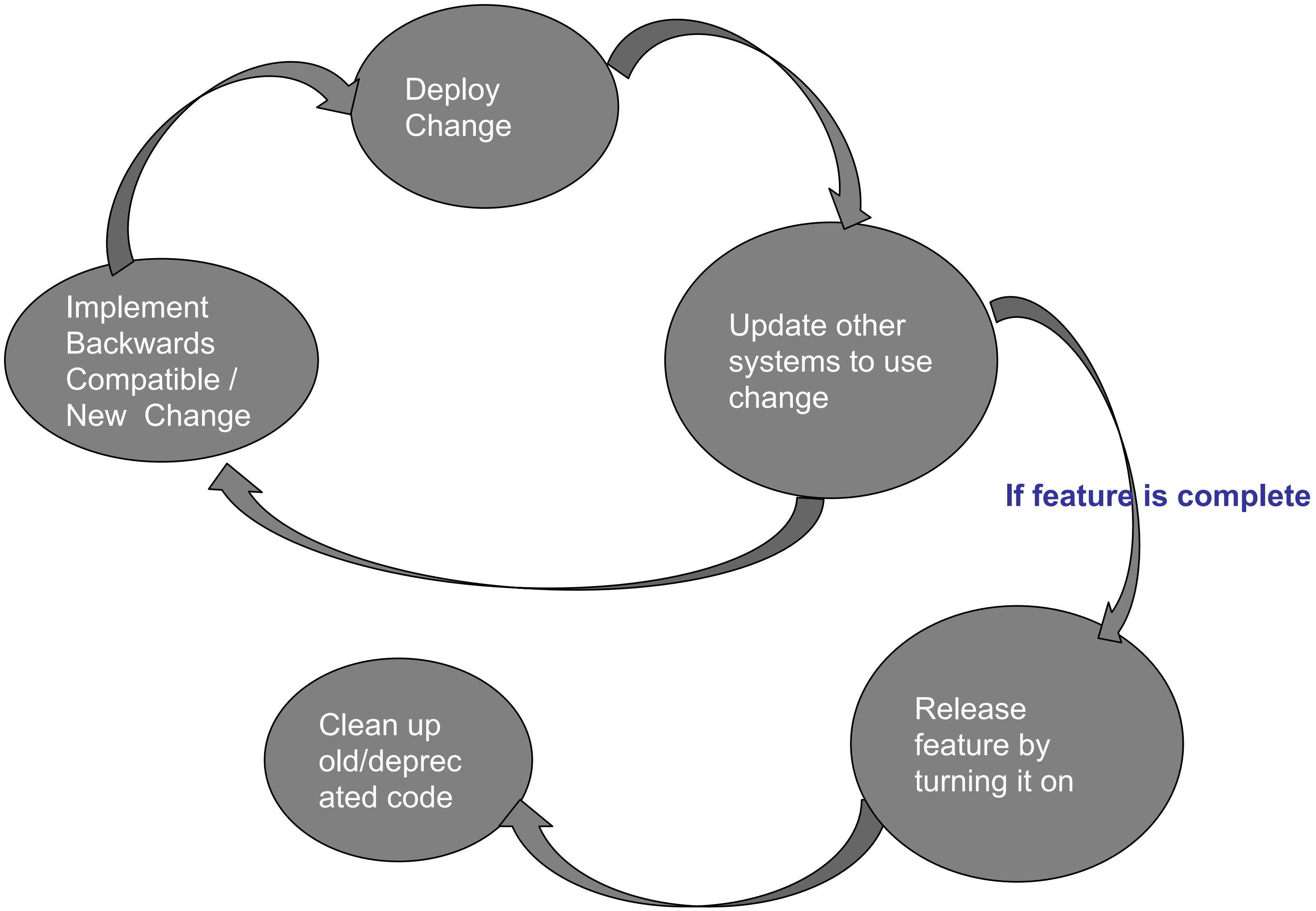
- Decoupling deployment and releases
- Backwards compatibility
- Versioning
- Feature Flags
- Clean up

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

– Albert Einstein

Decoupling Deployment and Releases

- Deploying code does not mean the releasing of functionality/features
- Do as small deployments as possible
- Will need change in thinking on how you go about coding a feature
- Keep on top of technical debt


Deploy Change
Update other systems to use change
If feature is complete
Release feature by turning it on
Clean up old/deprecated code
Implement Backwards Compatible / New Change

Feature Flags

- Feature flags are your friend
- Decouples deployments from releases
- Allows controlled roll out of new code
- Quick easy rollbacks without the need to do redeployments

Backwards Compatibility

- Wherever possible, update your code so that it works the old way alongside the new way.
- Tests will ensure that the old way is still working
- Loosen restrictions, tighten restrictions
- Temporary state to be in.

Versioning

- When it is impossible to do the change in a backwards compatible way, use versioning.
- Try to avoid as there is a lot of duplication and more clean up

Cleanup

- Most important part of this approach
- Prevents technical debt build up
- Have as part of definition of done
- Allows smooth change over to new code

Conclusion

Questions?