The Software Development Process at Amazon

Jonathan Weiss
Managing Director Amazon Web Services Germany GmbH

AWS OpsWorks, AWS Resource Groups, AWS Systems Manager
Amazon is hundreds of different businesses

AWS did over 1,430 major feature releases in 2017

No signs of slowing down
The Software Development Lifecycle

- **delivery pipeline**
  - build
  - test
  - release

- **feedback loop**
  - plan
  - monitor

- **developers**
- **customers**
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Let’s go back 15 years ago…
Monolith development lifecycle

developer

app

delivery pipeline

build

test

release
Service-Oriented Architecture (SOA)

Single-purpose Primitive

Connected w/ APIs

Highly decoupled

“Microservices”
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Two-pizza teams
Decentralized
Agility
Autonomy
Accountability
Ownership
“DevOps”
Missing Tools

developer | services | delivery pipeline

© 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Self-service
Technology-agnostic
Encourage best practices
Single-purpose services
Deployment service
No downtime deployments
Health tracking
Versioned artifacts and rollbacks
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Pipelines

Continuous delivery
Automated release process
Faster & more reliable releases
Adopted by virtually all teams
Microservice development lifecycle

developer

services

delivery pipelines

build → test → release

build → test → release

build → test → release

build → test → release

build → test → release

build → test → release

build → test → release

build → test → release
Thousands of teams
x Microservice architecture
x Continuous delivery
x Multiple environments

= millions of deployments
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Release Pipelines

Source → Build → Staging → Production

- Pull Source Code
- Run Build and Unit Tests
- Deploy to Staging Environment
- Deploy to Production Environment
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Release Pipelines

1. **Source**
   - Pull Source Code

2. **Build**
   - Run Build and Unit Tests

3. **Staging**
   - Deploy to Staging Environment
   - Deploy to One Box
   - Deploy to 1<sup>st</sup> AZ
   - Deploy to 2<sup>nd</sup> AZ
   - Deploy to 3<sup>rd</sup> AZ

4. **Production**
   - Deploy to 3<sup>rd</sup> AZ
Release Pipelines

Source -> Build -> Staging -> Production

- Pull Source Code
- Run Build and Unit Tests
- Deploy to Staging Environment
  - Functional Tests
  - Load / Perf Tests
  - Other Tests
- Deploy to One Box
- Deploy to 1st AZ
- Deploy to 2nd AZ
- Deploy to 3rd AZ
- Deploy to 1st AZ
- Deploy to 2nd AZ
- Deploy to 3rd AZ
Release Pipelines

Source
- Pull Source Code

Build
- Run Build and Unit Tests
- Deploy to Staging Environment
  - Functional Tests
  - Integration Tests
  - Load / Perf Tests
  - Other Tests

beta
- Deploy to Staging Environment
- Deploy to One Box
- Deploy to 1st AZ
- Deploy to 2nd AZ
- Deploy to 3rd AZ

gamma
- Deploy to Staging Environment
- Deploy to One Box
- Deploy to 1st AZ
- Deploy to 2nd AZ
- Deploy to 3rd AZ

Production
- Deploy to 1st AZ
- Deploy to 2nd AZ
- Deploy to 3rd AZ
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
The Edit-Compile-Test Loop
The Edit-Compile-Test Loop

ECT Loop
The Edit-Compile-Test Loop

ECT Loop

Code Review
The Edit-Compile-Test Loop

ECT Loop

Staging

Code Review
The Edit-Compile-Test Loop

- ECT Loop
- Production
- Staging
- Code Review
Catching Problems in the Development Lifecycle

ECT → Code Review → Staging → Production

Ideal Place  Great Place  Good Place  Last Resort
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
The Software Development Lifecycle

delivery pipeline

build | test | release

plan | monitor

feedback loop

developer(s)
customer(s)
Ops Culture

Ops Meetings

Ops Dashboards

- resource monitoring
- application monitoring
- canary monitoring
- client-side monitoring
- “order drop rate”
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
## Data Culture

### WBR Meetings

### Metrics Decks
- feature adoption
- segment growth
- funnel analysis
- cost structure
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Listen

It's easy to be right if you just listen to customers.

1. Ask them what they want
2. Get feedback on what you build
3. Course correct if need to
Feature vs. Need

Wenger 16999 Swiss Army Knife Giant
https://www.amazon.com/Wenger-16999-Swiss-Knife-Giant/dp/B001DZTJRQ
In the fine grained services approach that we use at Amazon, services do not only represent a software structure but also the organizational structure. The services have a strong ownership model, which combined with the small team size is intended to make it very easy to innovate. In some sense you can see these services as small startups within the walls of a bigger company. Each of these services require a strong focus on who their customers are, regardless whether they are externally or internally. To ensure that a service meets the needs of the customer (and not more than that) we use a process called "Working Backwards" in which you start with your customer and work your way backwards until you get to the minimum set of technology requirements to satisfy what you try to achieve. The goal is to drive simplicity through a continuous, explicit customer focus.

The product definition process works backwards in the following way: we start by writing the documents we'll need at launch (the press release and the fact) and then work towards documents that are closer to the implementation.

The Working Backwards product definition process is all about fleshing out the concept and achieving clarity of thought about what we will ultimately go off and build. It typically has four steps:

1. Start by writing the Press Release. Nall it. The press release describes in a simple way what the product does and why it exists - what are the features and benefits. It needs to be very clear and to the point. Writing a press release up front clarifies how the world will see the product - not just how we think about it internally.

2. Write a Frequently Asked Questions document. Here's where we add meat to the skeleton provided by the press release. It includes questions that came up when we wrote the press release. You would include questions that other folks asked when you shared the press release and you include questions that define what the product is good for. You put yourself in the shoes of someone using the product and consider all the questions you would have.

3. Define the customer experience. Describe in precise detail the customer experience for the different things a customer might do with the product. For products with a user interface, we would build mock ups of each screen that the customer uses. For web services, we write use cases, including code snippets, which describe ways you can imagine people using the product. The goal here is to tell stories of how a customer is solving their problems using the product.

4. Write the User Manual. The user manual is what a customer will use to really find out about what the product is and how they will use it. The user manual typically has three sections: concepts, how-to, and reference, which between them tell the customer everything they need to know to use the product. For products with more than one kind of user, we write more than one user manual.

Once we went through the process of creating the press release, facts, mockups, and user manuals, it is amazing how much clearer it is what you are planning to build. We'll have a suite of documents that we can use to explain the new product to other teams within Amazon. We know at that point that the whole team has a shared vision on what product we are going to build.
Ten Lessons Learned

1. Microservices
2. DevOps Teams
3. Self-Service Tools
4. Continuous Delivery
5. Pessimistic Deployments
6. Automated Testing
7. Optimize ECT Loop
8. Monitor Everything
9. Measure Everything
10. Listen to Customers
Thank you
Two reminders:

Please complete the session survey in the summit mobile app.