Software (and Software Pipeline) Architecture Matters

Anders Wallgren  |  CTO, Electric Cloud
@anders_wallgren
Agenda

I. Some context
II. Why software architecture is important
III. Loosely coupled architectures (microservices)
IV. Some patterns to help with microservices
V. Why software pipeline architecture is also important
VI. Resources
VII. Q&A?
Some Context
Every business is a software business

“Software is eating the world”

- FINSERV: building solutions to deliver better service
- RETAIL: building platforms for online sales and support
- AUTOMOTIVE: building services for the connected car
- FEDERAL: delivering on time and with higher quality
- HEALTHCARE: building solutions to improve care
- TELECOM: building embedded and online services
Software Is Eating My Bank

Software is now the primary driver of innovation & disruption.
What Is The One Question That Predicts Software Team Performance With Startling Accuracy?
“To what degree do we fear doing deployments?”

High-Performing IT Organizations Do It More Often

2017: 5x lower change failure rate, 96x faster recovery from failures.

From: IT Revolution and Puppet Labs’ 2016/2017 State of DevOps
But I was told this talk was about architecture?
Why Software Architecture is Important
Architecture is about the important stuff.

Whatever that is.

Martin Fowler’s definition
It is not the strongest of the species that survives, nor the most intelligent. It is the one that is most adaptable to change.

Charles Darwin

(sort of, see https://quoteinvestigator.com/2014/05/04/adapt/)
Software’s most important quality is its adaptiveness and ease of change.
Architectural outcomes: can my team…

• ...make large scale changes to the design of its system without the permission of someone outside the team, or depending on other teams?
• ...complete its work without fine-grained communication and coordination with people outside the team?
• ...deploy and release its product or service on demand, independently of other services the product or service depends upon?
• ...do most of its testing on demand, without requiring an integrated test environment?
• ...perform deployments during normal business hours with negligible downtime?
Loosely coupled architectures and teams are the strongest predictor of continuous delivery.
Loosely Coupled Architectures
(mostly Microservices, but other stuff, too)
Monolithic Architectures
The good, the bad, the ugly...

Pros
• Can be easier to develop
• Can be easier to test
• Can be easier to deploy

Cons
• Easier to produce spaghetti code
• Harder to integrate new technologies
• Harder to learn and understand the code
• You have to scale everything to scale anything
• Can’t deploy anything until you deploy everything
Microservices Architecture

A suite of services, each focused on doing one thing well

- Independently developed
- Independently deployable
- Exposes an API
- Runs in its own process

"Gather together those things that change for the same reason, and separate those things that change for different reasons."

– Robert Martin

Sounds loosely coupled to me!
What’s cool about Microservices?

- Divide and conquer complex distributed applications
- Independently developed and deployed, ideal use case for containers
- Freedom to choose the right technology (new or old) for each service
- Smaller more autonomous teams are more productive
Microservices are not a silver bullet!

Monolithic vs Microservices

@alvaro_sanchez
Get Loose! Some Best Practices For Decomposing The Monolith
Decomposing your monolith

- Identify what can be migrated to microservices - look for seams - areas of code that are independent, focused around a single business capability
- Use the strangler pattern liberally
- Domain-Driven Design (Domain Contexts, Aggregates) are useful
- You’ll probably get it wrong the first time
- Be Agile in your efforts - don’t boil the ocean
- Benchmark against the monolith early
- Don’t ignore organizational structure (Conway’s Law)
- Dependency analysis tools help, but are no panacea

“You don’t decompose monoliths, you *erode* them” - Mirco Hering
Decomposing your monolith’s state

• State may will be your largest source of coupling
• Understand your schema (if you have one)
  • Foreign key constraints
  • Shared mutable data
  • Transactional boundaries
• Is eventual consistency OK?
  • Avoid distributed transactions
  • Messaging between services
• Split data before you split code?
• Do you need an RDBMS at all or can you use other stores?
• Look into Saga, Event Sourcing, CQRS, but beware of complexity
Patterns To Assist in De-Coupling
Saga Pattern: Maintain data consistency across services

Problem:
Two services, each owns a business entity, we need a consistent update without 2PC

Solution:
Implement business logic that spans multiple services using service-local transactions + events

Each transaction publishes an event, which triggers the next transaction in the saga

Challenges:
• More complex, different programming model
• You may need to program compensating transactions to “undo”
• You need to be able to commit a transaction AND publish an event atomically (without 2PC)

http://microservices.io/patterns/data/saga.html
Command Query Responsibility Segregation

Problem: Two services, each owns a business entity, we need to query across both, without the benefit of JOIN

Solution: Treat the “CUD” and “R” of CRUD as separate concerns:

- **Command-side**
  - Handles create/update/delete
  - Emits events when data changes

- **Query-side**
  - Handles reads against data
  - Kept up to date by subscribing to data change events

Challenges:

- Complexity
- You need to be OK with eventual consistency, because lag
- Reporting databases may be a simpler approach

http://microservices.io/patterns/data/cqrs.html
**Event Sourcing**

**Problem:**
Atomically publish events when business data changes, without 2PC

**Solution:**
Persist business entities as a sequence of mutating events instead of (or in addition to) as a “row in the table”

**Benefits:**
- Audit log for free (hibernate-envers)
- You can query the state of the system at a point in time

**Challenges:**
- Complex and probably not something you build an entire system out of
- Queries are hideous, so this leads you in the direction of CQRS

http://microservices.io/patterns/data/event-sourcing.html
BFF: Backends For Frontends

• A variation on API gateways
• Tailored APIs for mobile devices, IoT, whatever
• “One backend per user-experience”
• “Impedance matching”
• Aggregate APIs to downstream services
• Use as strangler pattern to transition to new APIs

http://samnewman.io/patterns/architectural/bff/
Serverless

Backend-as-a-Service (BaaS)
• App depends largely or exclusively on third-party backend services
• Rich eco-system of services
• Why roll your own?

Function-as-a-Service (FaaS)
• AWS Lambda, Google Cloud Functions, Azure Functions, etc.
• Run backend code without managing server applications
• Code is triggered by events (S3 changes, messages, schedules)
• Easy horizontal scaling
• Very different programming model
How is “Serverless” not just PaaS?

Julz Friedman
@doctor_julz

if you think serverless is different than PaaS then either you or I have misunderstood what "serverless" or "PaaS" means

3:16 PM - 26 May 2016

adrian cockcroft
@adrianco

If your PaaS can efficiently start instances in 20ms that run for half a second, then call it serverless. twitter.com/doctor_julz/st...

5:43 AM - May 28, 2016

10 171 223
Serverless? Yes, but...

I mean how hard can it be to just glue together APIs that other people have written and support and scale? 😏 serverless #NoDev

Charity Majors
@mipsytipsy
12:48 PM - 23 May 2016

Figure out what your core differentiators are, and own the shit out of those
- Charity Majors

Caitie McCaffrey
@caitie
12:38 PM - 26 May 2016

Real question how does state get persisted with serverless?

I understand scale out of stateless servers, but who stores the state?

https://charity.wtf/2016/05/31/operational-best-practices-serverless/
A Few Other Things That Might Not Appear To Be Software Architecture, But Really, They Are
The Importance of Monitoring

Marius Ducea Retweeted

**Honest Status Page** @honest_update · Oct 7

We replaced our monolith with micro services so that every outage could be more like a murder mystery.

<table>
<thead>
<tr>
<th>RETWEETS</th>
<th>FAVORITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,303</td>
<td>1,003</td>
</tr>
</tbody>
</table>

4:10 PM - 7 Oct 2015 · Details
Monitoring Best Practices

• All services should log and emit monitoring data in a consistent fashion (even if using different stacks)
• Log early, log often
• Aggregate monitoring and log data into a single place
• Use techniques like correlation ids to track requests through the system
  • “So then requestId 0xf00dfee8 in the log on ms-app-642-prod becomes messageId 1125f34c-e34e-11e2-a70f-5c260a4fa0c9 on ms-route-669-prod?”
• Monitor latency and response times between services
• Understand what a well-behaving service looks like, so you can tell when it goes wonky
• Monitor the host (CPU, memory, etc)
Architect for Automated Testing

- Spend the time where you need but no more than necessary
- Use service virtualization / stubbing / mocking
- Consider using separate pipelines for end-to-end tests
- Performance testing at the service level is more important than in a monolith
- As always, flaky tests are the devil
Why Software Pipeline Architecture Matters

or

“If a tree falls in the woods, and nobody is around to hear it, and it hits a mime, does anyone care?” - Gary Larson
Software Delivery has changed...

Large App → Few Releases

Small & Modular App(s) → Many Releases
Many architectures, many pipelines, many destinations

“Bi-modal IT is a massive reductionist oversimplification” – Jez Humble
Different Flavors of Pipelines

**COMMIT PIPELINE [N]**
- DEV
- BUILD
- TEST

**COMMIT PIPELINE [1]**
- DEV
- BUILD
- TEST

**RELEASE PIPELINE [“N”]**
- INTEGRATION
- PRE-PROD
- PROD

### Build/Test Automation
- Build
- Unit / Other Tests
- ... 
- Publish Artifact
- Publish Artifact

### Cloud Resource Management
- Provision
- Configure
- Deploy
- Deploy (1...N)
- Provision
- Provision
- Provision
- Deploy (1...N)

### Deployment Automation
- Functional, Other Tests
- ... 
- Functional, Other Tests
- ... 
- Functional, Other Tests
- ... 

### Application Release Automation
- Smoke, Other Tests
- ... 
- Smoke, Other Tests
- ... 
- Smoke, Other Tests
- ...
Some Software Pipeline Best Practices
Have an artifact repository
Version everything (applications, infrastructure and middleware)
Automate (and Smoke Test) Deployments

“The most powerful tool we have as developers is automation” - Scott Hanselman

From: The DevOps Handbook, Gene Kim, Jez Humble, Patrick Debois, John Willis
Enable Self-Service Deployments
Allow teams to predictably and efficiently deliver application versions, on-demand.
Enable Self-Service Environment Provisioning, De-Provisioning

Spin up production-like environments anywhere in the cycle. Tear them down when finished.
Ensure Security and Auditability
Approvals and permissions ensure teams can deliver quickly and in compliance.
Best Practices for Software Pipelines

- Automate all the things – no more manual handoffs
- Your software pipeline needs to be as available as your app (HA, DR, etc.)
- Don’t tie your software pipeline architecture to your software architecture
- Are your tests automated? Really automated?
- Reduce onboarding time, waiting, and complexity with self-service automation/ChatOps approaches
- Provide a real-time view of all the pipelines’ statuses and any dependencies or exceptions
- Plug the pipeline into monitoring so that alerts can trigger automatic processes such as rolling back a service, switching between blue/green deployments, scaling and so on (and shift-left monitoring)
- Allow for automatic and manual (if you have to…) approval gates into and out of pipeline stages
- Create reusable models/processes/automation for your various pipelines
Simple Container Release Pipeline

1. Git Checkout
2. Code Scan
3. Docker Build
4. Unit Test
5. Publish to Docker Registry

1. checkTwistlockReport
2. Review Security Vulnerabilities
3. Deploy MotorBike StoreFront to Kubernetes
4. Test Automation
5. Monitoring - Dynatrace

Automated or Manual Approval Gates
Resources

- Migrating to Microservices at Netflix
  https://www.infoq.com/presentations/migration-cloud-native
- How we ended up with microservices
  http://philcalcado.com/2015/09/08/how_we Ended_up_with_microservices.html
- Confusion in the Land of the Serverless
  https://www.infoq.com/presentations/serverless-issues
- How to Make the Leap: Building Cloud-Ready Applications into the Architecture
  https://www.infoq.com/articles/cloud-ready-applications
- Backend for Frontend
  http://samnewman.io/patterns/architectural/bff
- WTF IS OPERATIONS? #SERVERLESS
  https://charity.wtf/2016/05/31/wtf-is-operations-serverless
  https://charity.wtf/2016/05/31/operational-best-practices-serverless
- Saga: How to implement complex business transactions without two phase commit
  https://blog.bernd-ruecker.com/saga-how-to-implement-complex-business-transactions-without-two-phase-commit-e00aa41a1b1b
- Developing Transactional Microservices Using Aggregates, Event Sourcing and CQRS
  https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson
  Mobile Needs A Four Tier Engagement Platform
  https://go.forrester.com/blogs/13-11-20-mobile_needs_a_four_tier_engagement_platform
- How to Make the Leap: Building Cloud-Ready Applications into the Architecture
  https://www.infoq.com/articles/cloud-ready-applications
- What Is "Cloud-Native" Data and Why Does It Matter?
  https://www.infoq.com/articles/cloud-native-data
- Low-risk Monolith to Microservice Evolution
  http://blog.christianposta.com/microservices/low-risk-monolith-to-microservice-evolution
Thank You!

Anders Wallgren  |  CTO, Electric Cloud
@anders_wallgren
SAY CLOUD
ONE MORE TIME

SAY "INTERNET OF THINGS"
ONE. MORE. TIME.

SAY "CLOUD-BASED" ONE MORE TIME
I DARE YOU