SAP Community

From Monolith to Microservices

Kai Spichale, SAP SE
July 5th, 2018
Disclaimer

The information in this presentation is confidential and proprietary to SAP and may not be disclosed without the permission of SAP. Except for your obligation to protect confidential information, this presentation is not subject to your license agreement or any other service or subscription agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or any related document, or to develop or release any functionality mentioned therein.

This presentation, or any related document and SAP’s strategy and possible future developments, products and or platforms directions and functionality are all subject to change and may be changed by SAP at any time for any reason without notice. The information in this presentation is not a commitment, promise or legal obligation to deliver any material, code or functionality. This presentation is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. This presentation is for informational purposes and may not be incorporated into a contract. SAP assumes no responsibility for errors or omissions in this presentation, except if such damages were caused by SAP’s intentional or gross negligence.

All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, which speak only as of their dates, and they should not be relied upon in making purchasing decisions.
Speaker: Kai Spichale

- Software architect for the SAP Community
- Agile software development and modern software architectures
- Specialized in Java and open-source technologies
- @kspichale
Agenda

What is the SAP Community?
Microservices Architecture of the SAP Community
Why Self-contained Systems?
Integration
  UI Integration
  Messaging
  API
SAP Cloud Platform
About
SAP
Community
SAP Community – The Next Generation

Enable anyone to
- Collaborate
- Engage
- Network

Access Community Content
- Includes blogging, discussions, documents
- Very rich user profile, including user content, activity
- Subscription system to follow users + content
The SAP Community has

- ~400,000 unique visitors per day
- ~120,000 active members
- Between 2,000 and 4,000 active members per day
- Millions of SAP professionals per year depend on getting answers to their questions
New Microservices Architecture
Architectural Overview

Apache Kafka (Messaging System)
Why Microservices?
Reason: Increase Modularity to Scale Development With Multiple Scrum Teams

- Increase modularity
  - No monolithic UI
  - Independent deployment units
  - Managed dependencies
  - Well defined APIs

Source: https://twitter.com/simonbrown/status/847339104874381312
Reason: Isolating Technical Decisions

- Teams work independently on the technical decisions and coordination
Reason: Increase Resilience

- Resilience is the ability of a system to adapt or keep working when challenges occur

**Scalability:** Start additional process on SAP CP

**Failure isolation:** Other apps keep running

**Evolvability:** Add new features independently

**Fault-tolerance:** HA messaging system
Challenges
¯__(ツ)__(¯
Assembling software from independent systems

- **Pros**
  - Avoid monolith that is hard to scale and extend

- **Cons**
  - New distributed system becomes very complex
  - Team wants to reduce complexity and coupling
Example: People App

- People App has a monolithic UI that consumes back-end microservices

Technical decisions and coordination between teams
Example: Editorial Content Service

- One possible approach to implement the new service could look like this:

  Technical decisions and coordination between teams
Example: Notification UI

- Notification UI is part of Activity Stream UI
Levels of Architectural Decisions

- The system has two separated levels of architectural decisions:
  1. Macro Architecture: Decisions that cover all modules
  2. Micro Architecture: Decisions taken for individual modules

- Goal:
  1. Minimal Macro Architecture
  2. Stable Macro Architecture

- Solution: Self-Contained Systems
Self-Contained Systems
What is a Self-Contained System (SCS)?

- Autonomous web application
  - E.g. autonomous Tomcat deployment on SAP CP
- Has its own business logic, own data storage, and own web interface
  - Fulfills its primary use case on its own
  - Minimal dependencies to other systems
- Owned by one team
- Communication with other systems is asynchronous wherever possible to improve performance and resilience
  - E.g. lightweight messaging with Apache Kafka
- Can have optional service API to share business logic
  - Should be well defined REST API
- Provides its features to end-users via its own UI
  - To reduce complexity and coupling

Source: http://scs-architecture.org
Logically related business functions are grouped in single SCS
- Responsible team can work independently on the technical decisions and coordination
- Limits dependencies to other requirements that are not related from a business perspective
- Team focuses on micro architecture (not macro architecture)
Integration

UI Integration

Messaging

APIs
Integration
UI Integration
With Hyperlinks
Use Hyperlinks to Navigation Between Systems
Use Hyperlinks to Navigation Between Systems (cont’d)
Use Hyperlinks to Navigation Between Systems (cont’d)
Integration

UI Integration

With Transclusion
Technical Options For Transclusion

- **Server-side**
  - Edge Side Includes (e.g. Akamai, Varnish)
  - Server-side Includes (e.g. Apache httpd)

- **Client-side**
  - In the browser via Ajax
Adobe Experience Manager (AEM)

- CMS for e.g. content topic pages, user homepage
- Main components:
  - Author module
  - Publish module
  - Dispatcher module
- Akamai = Content Delivery Network
Technical Options For Transclusion (cont’d)

Caching

Ajax

ESI

SSI

Loadbalancer

Dispatcher

Dispatcher

Publisher

Publisher

Author

AEM

Caching

Caching

Caching

Akamai

Microservices

direct connection to microservices

Publish connects to Microservices

Microservices
Technical Options For Transclusion (cont’d)

Server-Side Includes

- User agnostic data
- No Ajax
- 1x HTML page
- 1x cache control header
Technical Options For Transclusion (cont’d)

Ajax

- User agnostic data can be cached on Dispatchers
- Cached with different TTLs
Technical Options For Transclusion (cont’d)

Ajax

Cross-origin Ajax
- User-specific data
- Not cached on Dispatchers
- Only ~5% of traffic
Integration

Messaging
Messaging: Apache Kafka

- Messaging Systems connects most Microservices
- Adding a new Microservice is a small change (stable macro architecture)
- No need for service discovery
- Load balancing
- Based on [http://activitystrea.ms/specs/json/1.0/](http://activitystrea.ms/specs/json/1.0/)
A message has 1 producer and N consumers

Asynchronous communication: producer is not blocked by consumers

No message is lost if a consumer goes down (messages are stored for several weeks)
Not all connected applications are under our control, so we added push and pull services
Integration
API
HTTP API

Design
- Use of resources (not just a singular service endpoint) + consistent URI design
- Use of HTTP verbs + comply with semantics defined in RFC 7231
- No hypermedia controls (REST constraint) => Remote Procedure Calls
- No content negotiation
- Backward-compatible changes with JSON when possible
- Version number in URI

Implementation
- Spring Web MVC 5, Apache CXF
- Documentation with Swagger and Spring REST Docs
  - Interactive Swagger UI
  - Versioned HTML documentation in maven repository
Integration

UI Integration

Messaging

APIs
Technical Foundation
The SAP Cloud Platform (Neo)

- PaaS
- Support for HANA DB
- SAP UI 5

Multi-Cloud and Cloud Foundry

- Freedom to choose underlying cloud infrastructure
  - Data centers hosted by SAP and those run by partner IaaS providers: Microsoft Azure, Amazon Web Services, Google Cloud Platform
The SAP Cloud Platform (Neo)

17 Java Applications
- Java 8
- Java Web Tomcat 8 Runtime
- Using Spring, JPA, Liquibase and much more…

4 HTML5 Applications
- SAPUI5
- AngularJS

2 HANA Database Systems
- Split based on application size
- One schema per application
One Sub-Account per Operational Scenario

Sandbox
- Developer playground (e.g. for prototypes)

Development
- UI integration tests

QA
- Manual tests

Staging
- Performance tests

Production
- What you are using…
Version-controlled Configuration

Inter-service communication
- Managed in SAP CP Destinations

Landscape-specific configurations
- Different URLs
- Different proxy types (e.g. for Cloud Connector)
- Feature switches
Continuous Integration

- Separate builds for all microservices
- Dedicated Bamboo build servers
- Start builds upon commits
- Run automated unit and integration tests
- Automated multi-stage deployments
- Regular performance tests
- On-duty person & shared responsibility
Automated Deployment on all Landscapes

Pipeline-based Deployment
- One pipeline per micro-service & scenario

Development Landscape
- Only deployed if static checks & unit tests successful

QA Landscape
- Deployment only possible if integration tests successful

Production Landscape
- Deployment done after sprint results were accepted
Central Development & Operations Dashboard

Development
- Failed build pipelines
- Static code check results
- Test coverage
- Statistics over last 2 months

Operations
- Monitors QA, staging, production
- Runtime status
  - SAP CP (using monitoring API)
  - SAP Monsoon
  - AWS
- Health checks
- Statistics over last day
Thank you.

SAP Community
Kai Spichale kai.spichale@sap.com