Patterns and Pains of Migrating Legacy Applications to Kubernetes

Josef Adersberger & Michael Frank, QAware
Robert Bichler, Allianz Germany

@adersberger @qaware
Let's bring all our web applications onto a cloud native Platform
Digitalization
=> Agile
=> Cloud Native Platforms
Priorities:
(1) Time (1.5 years)
(2) Ops cost savings
(3) Migration costs
WE WERE BRAVE
WE FELT PAIN
WE DISCOVERED PATTERNS
All 152 legacy applications migrated and in production within 17 months.
All security-hardened and modernized to containerized 12-factor-apps.
Benefits leveraged: strong business case, higher availability, more agile teams.

WE WERE SUCCESSFUL
The Architect’s Point of View
Patterns for success
The Cloudalyzer

- EAM TOOL
- SYSTEM PROPERTIES
- QUESTIONNAIRES
- BASIC TOUR-DE-MIGRATION
- JIRA
- MIGRATION TASKS
- XLS
- STATIC ANALYSIS
- IBM MIGRATION TOOL
- QVALIDATOR
- SONARQUBE
- OWASP Scanner
- jQAssistant

ONE SINGLE SOURCE OF TRUTH

MIGRATION DATABASE

Tableau analysis
Questionnaire: Typical questions

- **Technology stack** (e.g. OS, appserver, jvm)
- **Required resources** (memory, CPU cores)
- **Writes to storage** (local/remote storage, write mode, volume)
- **Special requirements** (native libs, special hardware)
- Inbound and outbound **protocols** (protocol stack, TLS, multicast, dynamic ports)
- **Ability to execute** (regression/load tests, business owner, dev knowhow, release cycle, end of life)
- **Client authentication** (e.g. SSO, login, certificates)
Emergent design of cloud native software landscapes
Architecting hundreds of applications

- Application Blueprint: Describing target architecture and some rules & principles
- Migration Cookbook: Guidance on how to migrate the applications based on the application blueprint. Single source of truth & know-how externalization
- Tour-de-Migration: Visiting all applications and collect open issues
- GoLive Readiness Checklist: Criteria to be checked before GoLive
1) how to enhance cloud nativeness?

2) how to cut the monolith?

3) how to obtain an identity token?
1) how to enhance cloud nativeness?
2) how to cut the monolith?
3) how to obtain an identity token?
A sweet spot for legacy apps

Put the monolith into a container: do not cut, do not enhance with features in parallel

... and enhance the application according the 12 factors

Cloud Friendly Apps

- Containerization
- 12-Factor App Principles

- Monolithic Deployment
- Traditional Infrastructure

- Microservices
- Cloud-native Apps
Sidecars to the rescue
Container patterns applied

**Sidecar:** Enhance container behaviour
- Log extraction
- Task scheduling

**Ambassador:** Proxy communication
- mTLS tunnel
- Circuit Breaking
- Request monitoring

**Adapter:** Provide standardized interface
- Configuration (ConfigMaps & Secrets to files)

1) how to enhance cloud nativeness?

2) **how to cut the monolith?**

3) how to obtain an identity token?
Anti-pain rule: Don’t cut the monolith
Anti-pain rule: Don’t cut the monolith

Before:
- Clients
- Monolith
- Backend

After:
- Clients
- Some magic sauce
- Security gateway
- Monolith
- Backend
1) how to enhance cloud nativeness?

2) how to cut the monolith?

3) how to obtain an identity token?
Security service to the rescue

BEFORE

CLIENTS

MONOLITH

BACKEND

AFTER

CLIENTS

SECURITY SERVICE

SECURITY GATEWAY

MONOLITH

BACKEND

Adapting multiple authentication mechanisms to a uniform OIDC token.
Initially we thought we’ll run into k8s restrictions on our infrastructure like:

- No support for multicast
- No RWX PVC available

We did. But all required refactorings were moderate effort and lead to a better architecture.
The Lead Developer’s Point of View
The almighty legacy framework

• “worry-free package framework” from the early 2000s with about 500kLOC, 0% test coverage and multiple forks

• Strategies:
  • the hard way: consolidate forks and migrate manually and increase coverage
  • decorate with ambassadors, sidekicks and adapters
  • do not migrate parts and replace that API within the applications

APPLICATION

ALMIGHTY LEGACY FRAMEWORK

J2EE 1.4 APPSERVER
JVM 1.6

• from J2EE 1.4 to JEE 7 and Java 6 to 8
• add identity token check and relay
• modify session handling (synchronization)
• modify logging (to STDOUT)
• modify configuration (overwrite from ConfigMap)
• enforce TLS 1.2
• place circuit breakers
• predefined liveness and readiness probes
TIME-OUTS
Timeouts: The pain

• Kinds

  - causes bad user experience
  - hurts the stability of your entire cloud

  • Unable to distinguish errors from legitimate waits
  • Diminishes self healing capabilities
  • Promotes cascading failures
Timeouts: The pain

• Kinds

  • Timeout often too high. This ...
    – causes bad user experience
    – hurts the stability of your entire cloud
      • Unable to distinguish errors from legitimate waits
      • Diminishes self healing capabilities
      • Promotes cascading failures
Timeouts: Recommendations

- Keep timeouts within the following ranges
  - 1-3s for getConnection & connect
  - 3-60s for socket/read - aim as low as possible
  - 1-3min for TTL/KeepAlive of pooled connections
    - Allow for dynamic DNS changes and dynamic scaling of backend services
    - Tradeoff between reaction time and performance

- Cascade timeouts
  - outer layer highest
  - inner layer lowest
Latency

• Pain: Dramatic increase in latency
  You can't scale away latency!
    – Every layer and new infrastructure component adds processing time
    – Everything TLS1.2 secured adds processing time
    – Physical distance: Cloud -> OnPrem

• Heaviest impact on n+1 patterns in applications
  – Adjust batch/fetch size
  – Parallel fetch
  – Ultima ratio: on prem (lightweight) service layer close to DB

• General
  – Performance experts in support team
  – Caching
  – Use diagnosability tools...
Latency

• Pain: Dramatic increase in latency
  You can't scale away latency!
  – Every layer and new infrastructure component adds processing time
  – Everything TLS1.2 secured adds processing time
  – Physical distance: Cloud -> OnPrem

• Heaviest impact on n+1 patterns in applications
  – Adjust batch/fetch size
  – Parallel fetch
  – Ultima ratio: on prem (lightweight) service layer close to DB

• General
  – Performance experts in support team
  – Caching
  – Use diagnosability tools...
DIAGNOSABILITY
1. **Early on** - diagnose cloud platform issues upfront

2. **Holistic** - monitor and correlate everything (infrastructure & apps, multiple levels, metrics & logs & traces)

3. **Mandatory** - everyone has to use it

4. **Automatically** - auto-instrumentation not involving devs
- High effort to instrument for valuable insights
- Scalability unclear for hundreds of applications
- Applications have no time to run their own Prometheus instance

- Scalability unclear (a lot of events lost)
- Applications have no time to run their own EFK instance
- Non-standardized log format requires custom log rewrite adapter but no fluentd DaemonSet

**Grafana**

**Prometheus**

**Metrics**

**Application Diagnosability?**

**Traces**

- Scalability unclear for hundreds of applications (Jaeger & ZipKin)
- Applications have no time to run their own instance

**Events / Logs**

- Fluentd
Want to move fast? Buy first, reduce cost later

Metrics

Application Diagnosability

... use APM tools like Dynatrace and Instana

Traces

Events / Logs
SESSION STATE
Session state

1. Session Stickiness: not within the cloud!
2. Session Persistence
   • Existing DB: perf impact to high 😞
   • Redis: no TLS out of the box and infrastructure required 😞
3. Session Synchronization
   • App-Server: no dynamic peer lookup within k8s 😞
   • Hazelcast: TLS only in paid enterprise edition 😞
   • ...

☹️ 表情符号
Session synchronization with Ignite

• Apache Ignite as in-memory data grid
  – Embedded within application or standalone (in sidecar)
  – Cumbersome but working k8s peer lookup

• Look out for ...
  – Java serialization
  – Legacy frameworks with custom session handling
  – Prevent generating sessions for e.g. health check requests
  – Applications putting large things into the “session” and misuse session as cache
#@!!#@$
## Other technical pain points

<table>
<thead>
<tr>
<th>Pain</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy crypto without TLS 1.2 and SNI support (e.g. Java 1.6)</td>
<td>- Find matching cipher suites</td>
</tr>
<tr>
<td></td>
<td>- Add a security proxy</td>
</tr>
<tr>
<td>Legacy apps violating HTTP standards</td>
<td>Refactor</td>
</tr>
<tr>
<td>Access source URLs in redirect loops (e.g. IDP login)</td>
<td>Use x-forwarded header and provide according filter</td>
</tr>
<tr>
<td>No automated test suites</td>
<td>- Automated high-level tests</td>
</tr>
<tr>
<td></td>
<td>- Test generation (e.g. evosuite)?</td>
</tr>
</tbody>
</table>
The Project Manager’s Point of View
Patterns for success
Management support

- Strong management support
- Clear scope
- Courage to drive the change to cloud native development

Let's bring all our web applications into the cloud!
Project Marketing & Motivation

Identification & Celebration
Co-Location space

One LEAP-Area
- Support- & Industrialization team
- In case of required support: Migration team
Industrialization
DOZENS OF MIGRATION PROJECTS RUNNING IN PARALLEL
(organized in release trains)

- Training sessions
- Support sessions
- Co-Location & remote

SUPPORT TEAM
- Feedback

- Guidance / best practice sharing (cookbook, sample application)
- Unified development environment (via GitHub)
- Standard base images
- Pre-migrated frameworks
- Solutions: Security service, ambassadors

INDUSTRIALIZATION TEAM
- Application blueprint
- Migration database

ARCHITECTURE TEAM
Transparency & information radiators

App-Support

Activities & Milestones

GoLive Planning

Quality

Operational