Avaloq’s journey to a standardized, scalable banking reference architecture

Christoph Zehnder, Avaloq
Daniel Bejinaru, Avaloq
Daniel Schaefer, Red Hat

May 2019
Speakers

Christoph Zehnder
Avaloq
Hands on software architect

Daniel Bejinaru
Avaloq
From the "as a service" part of the company

Daniel Schaefer
Red Hat
Sr. Solutions Architect working with ISVs across EMEA
A software

- Core banking software, digital banking and digital wealth management used by 150 banks
- Offered "as a service" and "on premise"

A company

- 2200 employees (including 700 developers)
- Service centers in Switzerland, Germany and Singapore
- Development in Switzerland, UK, Philippines
Why are we here?
Our story

• OpenShift is a great product
• But challenging to implement in an enterprise / financial industries environment
• Present the challenges we faced and some decisions we took

And: Don’t expect us to sell our products or services!
But: We are always looking for good developers :)
Adapt to changed requirements
Avaloq’s open banking architecture

Browser-based / mobile UIs / Fintech
REST APIs
Business logic / Customization
Scalable low-latency read caches
Container platform
Standard cloud

The platform to efficiently and effectively develop and operate great banking functionality.
Container platform
Transformation in ecosystems

Why we think this case is interesting

It’s all about the application!
Boosting customer success with modernized applications and by enabling new DevOps methodology.

Standardize and open up!
Because solving complex business problems with complex software on a complex IT stack won’t scale well...

Ecosystem Collaboration!
ISV, Service Provider, Technology Partner and System Integrator collaborating for joined customers.

Who are you going to call?
Embedded Model allows Avaloq ownership of customer support experience with Red Hat in the back.

#avaloq #redhat #rhsummit
Challenges

OpenShift is a technical requirement
• Investment in implementing and supporting the Avaloq Container Platform
• Client uncertainty poses open questions (from design, security, to high effort estimation and budget)

Financial Regulations / Security
• Protecting Client Identifying Data
• Swiss banking rules are strict - there is no easy way to do "DevOps"
• Service provider and software builder for banks (not a bank)
Shift of responsibilities

“Dev” and “Ops” in two separate companies

Build trust
Focus on security, enable transparency and communication

Collaboration
Involvement and contribution

Designed for operations
New responsibilities for all parties

Reuse
And avoid re-inventing the wheel
Build Trust: Security
Secure container images

FROM docker.io/centos
ENV privatekey myprivatekey.pem
CMD while true; do sleep 1; done
Security pipeline

<table>
<thead>
<tr>
<th>ID</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7b4c28c77377db02</td>
<td>critical</td>
<td>Detect plaintext private keys in environment variables</td>
</tr>
<tr>
<td>7b4c28c77377db02</td>
<td>high</td>
<td>(CIS_Docker_CE_v1.1.0 - 4.1) Image should be created with a non-root user</td>
</tr>
<tr>
<td>7b4c28c77377db02</td>
<td>high</td>
<td>Image is not trusted</td>
</tr>
</tbody>
</table>
Secure container images

FROM docker.io/centos
ENV privatekey myprivatekey.pem
CMD while true; do sleep 1; done

- No trusted Avaloq Base Image
- Credentials in environment variables
- No unprivileged user specified
Security guidelines and best practices

Guidelines for handling secrets/credentials inside containers
- A container/pod accesses credentials using a dedicated "secret" object.
- Secrets have to be stored encrypted.
- Secrets have to be transmitted encrypted from the "vault" to the container.
- Access to the secrets has to be "access-controlled" and "revocable".
- **Secrets must not be available as Environmental Variables (risk of being logged)**
- **HIGHLY RECOMMENDED:** Access to secrets has to be audit/logged

Prevent Root Processes inside Containers / Prevent Privileged Containers
- Containers/pods are executed with the restricted SCC by default.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container Images and Build File</td>
<td>• Ensure a user for the container has been created (Either manually or automatically by OpenShift)</td>
</tr>
<tr>
<td></td>
<td>• Ensure that containers use trusted base images</td>
</tr>
<tr>
<td></td>
<td>• Ensure unnecessary packages are not installed in the container</td>
</tr>
<tr>
<td></td>
<td>• Ensure images are scanned and rebuilt to include security patches</td>
</tr>
<tr>
<td></td>
<td>• Ensure Content trust for Docker is Enabled (<a href="https://docs.docker.com/engine/security/trust/content_trust">https://docs.docker.com/engine/security/trust/content_trust</a>)</td>
</tr>
<tr>
<td></td>
<td>• Ensure setuid and setgid permissions are removed in the images</td>
</tr>
<tr>
<td></td>
<td>• Ensure secrets are not stored in Dockerfiles</td>
</tr>
<tr>
<td></td>
<td>• Ensure verified packages only are installed</td>
</tr>
<tr>
<td></td>
<td>• Setuid and Setgid binaries should also be removed from images, lessening the chance of privilege</td>
</tr>
<tr>
<td></td>
<td>• Container processes run as non-privileged USER</td>
</tr>
</tbody>
</table>
Security pipeline
Approved container images

Developer follow best practices

- Vulnerability Scanning
- Compliance Checks

Periodic rescanning of all images

Signature validation on OpenShift Nodes
What cluster setup do I need?
Does this fit into a secure 3-tiered network?
"It depends …"

Reference Architectures
- PCI-DSS Reference Architecture
- Ten Layers of Container Security
- OpenShift Docs

Physical vs. Logical Segregation
“Is SDN secure enough?”
Physical node per tenant?
Can Dev and Prod run in the same cluster?

Node Placement
Where to put Masters, Routers and Workers?
One cluster per network zone is more secure – but way more expensive…
Everything in one zone to reduce complexity?

Lock down
Will additional firewalls lead to more security?
Blog Post on Security Zone Coexistence Approach

#avaloq #redhat #rhsummit
OpenShift reference architecture

Considered the reference for the implementation

"It is not enough" to fulfill security requirements

Keep compatibility, avoid major customizations
Different placement options
OpenShift allows flexible placement of nodes in different zones

Source: Blog Post on Security Zone Coexistence Approach

#avaloq #redhat #rhsummit
How can this run in my datacenter?

Security
- Network segregation
- Reduced attack surface and impact mitigation

Operations
- Established Operating Model and tools with layered accountability
- Strictly defined processes, rules and tools from DEV to OPS

Control
- Strictly controlled configuration
- Traditional change and release management process
OpenShift reference architecture
Internet publishing

Untrusted Zone (DMZ)
- Internet facing components
Security zones

Untrusted Zone (DMZ)
- Internet facing components

Semi-trusted Zone
- Internet Exposed Services

Trusted Zone
- Customer Identifying Data (CID)
- Business critical processes
Security zones: two clusters

Untrusted Zone (DMZ)
- Internet facing components

Semi-trusted Zone
- Internet Exposed Services

Trusted Zone
- Customer Identifying Data (CID)
- Business critical processes
Impact on Customers and Service Centers

The answer is standardization, re-use and automation!
How can I efficiently deploy applications?
Different options to deploy containers

Deployments can be easily automated, choose your way:

- OpenShift Deployments & Templates
- Ansible Playbook Bundles
- Operator Framework

- Native Kubernetes commands (kubectl)
- Helm Charts
- Many open source tools, active and deprecated (ksonnet, ...)
- Anything can call an API (CI/CD tools, automation tools, scripts, ...)

#avaloq #redhat #rhsummit
Which containers do I have to deploy?
Define dependencies

manifest:
  version: 1.2.0
components:
  com.avalq:container-component-dep-example:
    version: 0.1.0-dev
    requires:
      components:
        com.avalq:avalq-jpa-provider: 2.1.0
        com.avalq:container-component-main-example:
          version: 0.1.0-dev
          requires:
            components:
              com.avalq:container-component-dep-example: 0.1.0
              com.avalq:session-manager-platform: 0.74.0
  acp-interfaces:
    session_ws: 1.0.0
  acp-stream-solutions-per-stream:
    R4.6:
    - 2289983
    R4.5:
    - 2289982
    R4.4:

#avaloq #redhat #rhsummit
Customer selects components

```plaintext
products:
- components:
  - group: com.avaloq
    name: avaloq-aws
    version: 2.24.0
  - group: com.avaloq
    name: aws
    requiredProducts: null
    version: 1.0.0
- components:
  - group: com.avaloq
    name: avaloq-zookeeper
    version: 3.0.0
  - group: com.avaloq
    name: avaloq-kafka
    version: 5.3.0
  - group: com.avaloq
    name: avaloq-meteor
```
Store everything in a repository

Constellation

.git
Add configuration

- Constellation
- Business Configuration
- Technical Configuration
- Secrets

#avaloq #redhat #rhsummit
Use a job to deploy to OpenShift

OpenShift configuration in source control
Why "do it yourself"?
How can I operate this?

What do I need additionally?

→ Dashboards
→ Alerts
Dashboards and alerts

Producer
(Developer / Vendor)

Consumer
(Service Center / Bank)

Container

Kubernetes Object Specs

Dashboards

Alerts

Dashboards (ConfigMap)

Alerts (ConfigMap)

Grafana

Prometheus

Watcher Loader
Conclusion

- There are many solutions with different options from Red Hat and the cloud native community.
- But Avaloq customers can’t lose time evaluating options.
- They just need a good default! (for the banking industry)
- Avaloq needs standardization to support our customers! (we are an ISV)
Avaloq Container Platform

Share within the Avaloq community
Concepts
Reference architecture
Security: threat model

- Spoofing Identity
- Tampering with data
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of privilege

<table>
<thead>
<tr>
<th>#</th>
<th>Threat</th>
<th>Further Explanations</th>
<th>External / Internal</th>
<th>Requirement(s) to prevent threat (bold indicates bare minimum requirements)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T201</td>
<td>An attacker (unauthenticated, authenticated but not authorized) injects a container image to the production registry that is not signed.</td>
<td>The attacker is unauthenticated and is not considered a &quot;valid&quot; user of the infrastructure. The attacker is authenticated, but not authorized to upload images to the production registry. Containers are considered as blocks of data in a container infrastructure. They contain the applications that are executed by this infrastructure. Bypassing the signature, an attacker can provide containers (data packages) that are not supposed to be executed. The content of a container image defines what is executed by the infrastructure. Being able to replace a container image enables an attacker to inject a malicious image and eventually execute malicious code on the container infrastructure.</td>
<td>Internal + External (the production registry is accessible from outside)</td>
<td>SEC-01, SEC-05, SEC-15, SEC-20, SEC-24, SEC-30, SEC-39, SEC-47, SEC-23, Sec-24, SEC-25, SEC-42, SEC-35</td>
</tr>
<tr>
<td>T103</td>
<td>The root user inside the container is not mapped to a non-root user on the host</td>
<td>The root user inside the container is mapped to the root user on the application node. Since these two users should be essentially different from each other, the wrong mapping makes them equivalent. A root user in the container has then the same identity as the root user on the host.</td>
<td>Internal + External (users of the container application can be internal and external)</td>
<td>SEC-09, SEC-63, SEC-76</td>
</tr>
</tbody>
</table>
Security guidelines and best practices

Guidelines for handling secrets / credentials inside containers
- A container / pod accesses credentials using a dedicated "secret" object.
- Secrets have to be stored encrypted.
- Secrets have to be transmitted encrypted from the "vault" to the container.
- Access to the secrets has to be "access controlled" and "revocable".
- Secrets must not be available as Environmental Variables (risk of being logged)
- HIGHLY RECOMMENDED: Access to secrets has to be audit/logged

Prevent Root Processes inside Containers / Prevent Privileged Containers
- Containers / pods are executed with the restricted SCC by default.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Best Practices</th>
</tr>
</thead>
</table>
| Container Images and Build File | • Ensure a user for the container has been created (Either manually or automatically by OpenShift)  
• Ensure that containers use trusted base images  
• Ensure unnecessary packages are not installed in the container  
• Ensure images are scanned and rebuilt to include security patches  
• Ensure Content trust for Docker is Enabled (https://docs.docker.com/engine/security/trust/content_trust/  
• Ensure setuid and setgid permissions are removed in the images  
• Ensure secrets are not stored in Dockerfiles  
• Ensure verified packages only are installed  
• Setuid and Setgid binaries should also be removed from images, lessening the chance of privilege  
• Container processes run as non-privileged USER |
string sInput;
int ilength, iN;
double dblTemp;
bool again = true;

while (again) {
    iN = -1;
    again = false;
    getline(cin, sInput);
    system("cls");
    stringstream(sInput) >> dblTemp;
    ilength = sInput.length();
    if (ilength < 4) {
        again = true;
        continue;
    } else if (sInput[ilength - 3] != '.') {
        again = true;
        continue;
    } while (++iN < ilength) {
        if (isdigit(sInput[iN])) {
            continue;
        } else if (iN == (ilength - 3)) {
            continue;
        }
    }
}
Example deployment

Anisble roles

Following Anisble roles are provided (also used by provided playbooks):

- **Provisioning**
  - `avaloq.apsr-aws`: Anisble AWS configuration
  - `avaloq.apsr-aws`: Anisble AWS services (AWS) status parsing and configuration generation
  - `avaloq.apsr-provision`: Anisble AWS services (AWS) infrastructure provisioning

- **Installation**
  - `avaloq.apsr-auth`: Anisble ACPR tech
  - `avaloq.apsr-docker`: Anisble ACPR Di
  - `avaloq.apsr-inventory`: Anisble ACPR
  - `avaloq.apsr-logging`: Anisble ACPR L
  - `avaloq.apsr-monitoring`: Anisble ACPR M
  - `avaloq.apsr-reporting`: Anisble ACPR installation prerequisites
  - `avaloq.apsr-ansible`: Anisble ACPR OpenShift repositories setup

- **Administration**
  - `avaloq.apsr-backup`: Anisble ACPR OpenShift backup and restore

---

---
Call to action

• Q&A
• We are looking for feedback and others interested in similar challenges
• Open sourcing is not yet an option for Avaloq
• Who can recommend a format for collaboration?