Microservices and software development infrastructure

Stefano Bovina

Agenda

Il target di questa infrastruttura

Brevi cenni sui microservizi

Sfide tecnologiche/tecniche

Panoramica della infrastruttura

Problematiche di sicurezza e non solo

Modifiche relative ai processi di sviluppo e gestione del software

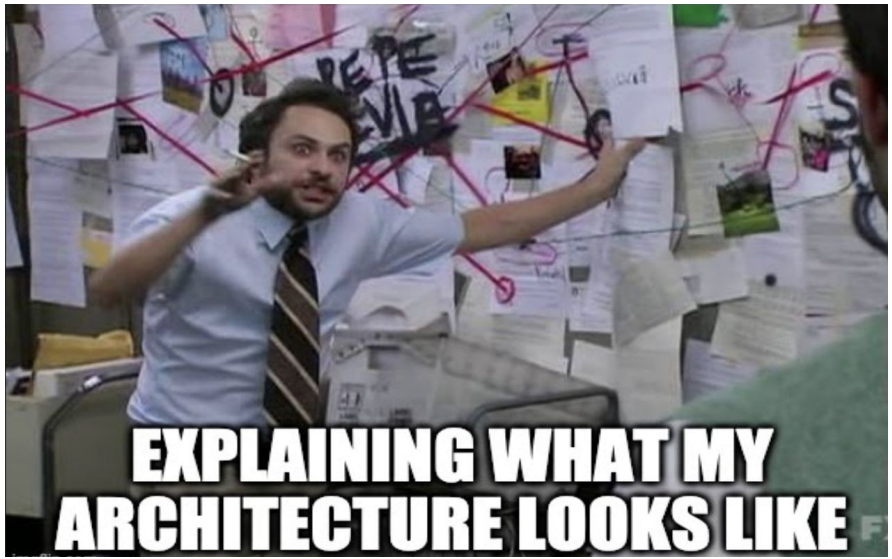
Considerazioni generali ed altre conseguenze

Stato dell’arte

Punti aperti

Scopo

- A supporto dei microservizi
- Evoluzione di quanto impostato da tempo
- A supporto della scrittura di software di migliore qualità
- Rinnovamento tecnologico
- Per risolvere problemi non risolvibili con le applicazioni legacy
- Adeguamenti relativi a security e compliance



Cosa dovrà ospitare?

Applicazioni sysinfo:

- nuovi servizi (Springboot/NodeJS)
- dipendenze che non necessitano di persistenza a lungo termine (es: cache)
- servizi esistenti (ora su VM): da riprogettare/riscrivere o migrare

E le applicazioni sysinfo di terze parti (es: librofirma) delle quali non abbiamo i sorgenti (black-box)?

- Meglio un approccio SaaS se possibile
- Installazioni on-premise dovranno essere valutate di volta in volta; se conformi ai nostri standard potranno girare su questa piattaforma

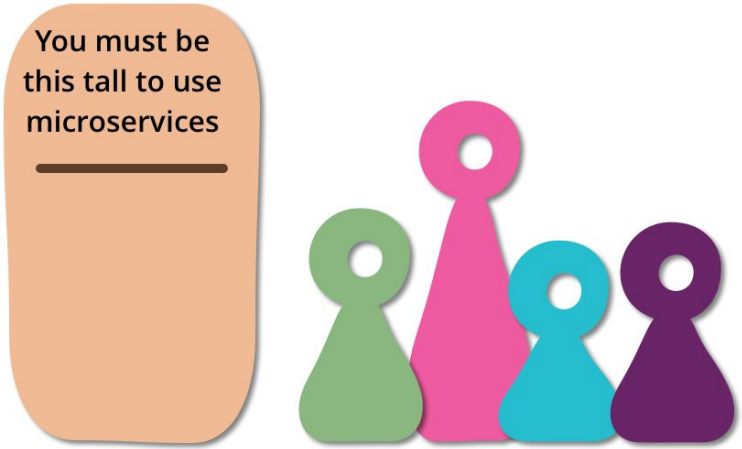
E i servizi da uso interno (es: Grafana ecc)?

- Verrà valutato di volta in volta a seconda del contesto e necessità tecniche

Microservices

- Rapid provisioning ✔
- Basic monitoring ✔
- Rapid application deployment ✔
- Devops culture ✓

“cultural movement based on the concept of breaking down barriers between dev and ops”



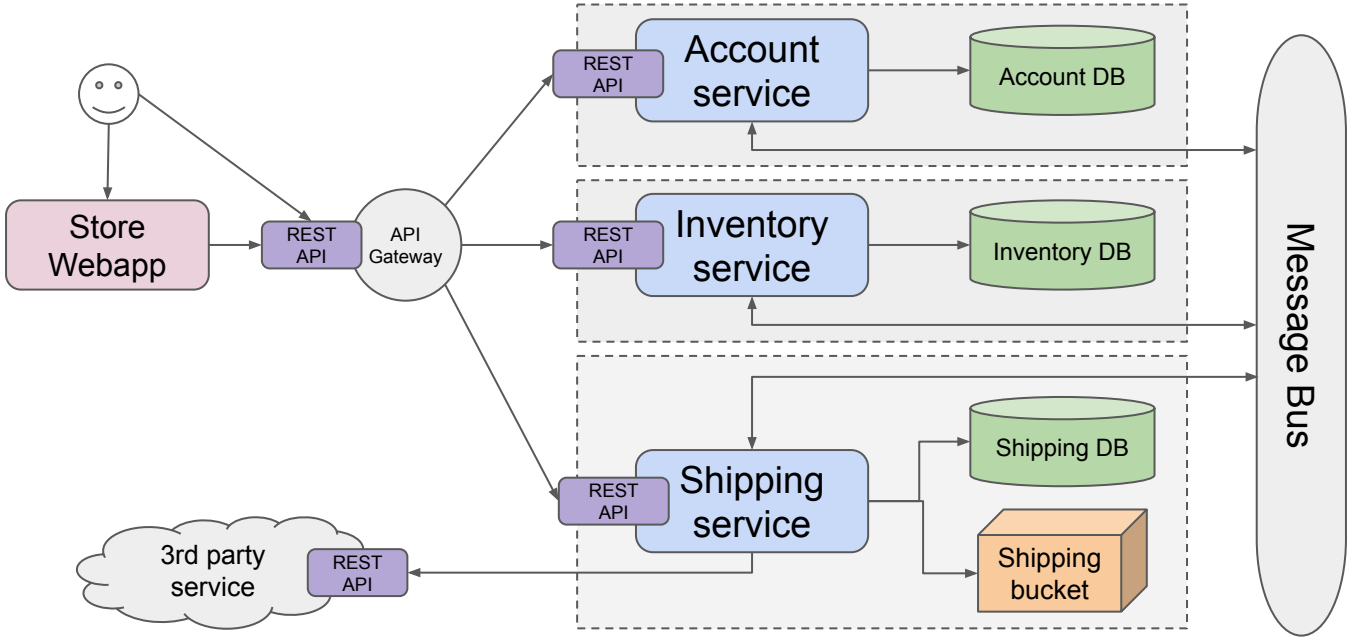
Non si discuterà se usare/non usare i microservizi ma riflettiamo su quanto segue ...

“Any organization that design a system will inevitably produce a design whose structure is a copy of the organization’s communication structure” (Conoway’s law, 1967)

“...so If you have a four groups working on a compiler, you’ll get a 4-pass compiler”

Microservices

*"...the microservice architectural style is an approach to developing a single application as a **suite of small autonomous services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery..."*



NOTA: lo schema è solo un esempio

Microservices

Pro:

- Strong Module Boundaries: Microservices reinforce modular structure
- Independent deployment
- Higher degree of organizational autonomy
- Technology Heterogeneity
- Optimized for replaceability
- Scaling independently
- Leads to Improved Fault Tolerance (if we understand and plan for failures)
- Ease of understanding of the codebase of the software system
- Isolation of data and isolation of processing around that data

Cons:

- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure
- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency (Multi-services transactions/changes are complex)
- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly
- Global automated testing is more complicated

Microservices: others core concepts/keywords

- Things that change together should be packaged together
    - Loosely coupled: a change to one service should not require a change to another (DRY mantra should not be strictly used with microservices: it can lead to high coupling)
    - High cohesion: a service should implement a small set of strongly related, well defined purpose, functions
    - **if a collection of microservices is often being changed together, probably you are building a distributed monolith**
- Avoid breaking changes (don’t break consumer contracts)
- Hide implementation details (Information hiding): allow one service to evolve independently of another
- Avoid shared databases/schema => Multi schema Join/View/Foreign Key cannot be used
- Prefere Async communication (use dumb-middleware with smart endpoints)
- Distributed system => design for failure
- Automate everything
- Test your software
- Products not projects

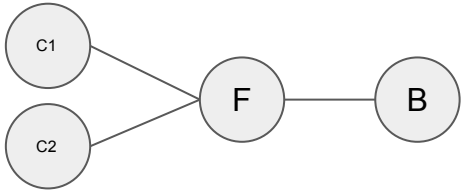
Distributed computing fallacies and other requirements

Fallacies

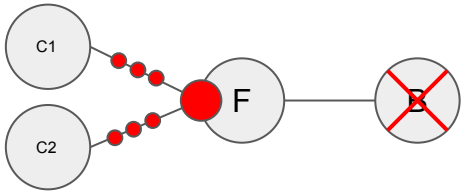
Fallacies/requirements	Solutions
The network is reliable	Circuit breaker (Resilience4j), retry and timeout design pattern (Resilience4j), message queues (Kafka)
Latency is zero	caching strategy (Redis), bulk requests, placement/affinity policy (see Kubernetes policy)
Bandwidth is infinite	throttling policy, small payloads
The network is secure	firewall (network policy/micro segmentation), encryption (mTLS, Cert-manager), AuthN/AuthZ (OIDC/Oauth2)
Topology doesn't change	no hardcoded IPs, service discovery tools (see Kubernetes service discovery)
There is one administrator	DevOps culture
Transport cost is zero	standardized protocols like JSON, cost calculation
The network is homogeneous	Circuit breaker (Resilience4j), retry and timeout design pattern (Resilience4j)
Observability	Monitoring system (Prometheus/Grafana/Sensu/InfluxDB), Log aggregation (ELK)
Automation culture	CI/CD platform (Gitlab, ArgoCD), IaC paradigm (Helm/Kustomize/Puppet)
Secret management	Vault

Circuit breaker: the problem

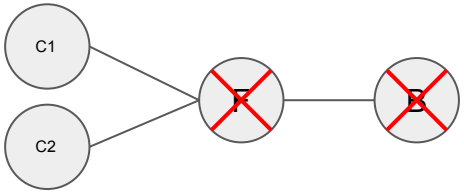
Local failures can propagate all over your architecture and destroy all your systems



- C makes a request to F
- F makes a request to B
- After short time Client got a response



- If B:
  - is down
  - is up but the network is unreachable (network partition)
  - is up but is very slow
- C makes a request to F
- F need to wait until the timeout before came back to C
- During the timeout period requests pile up on the F service

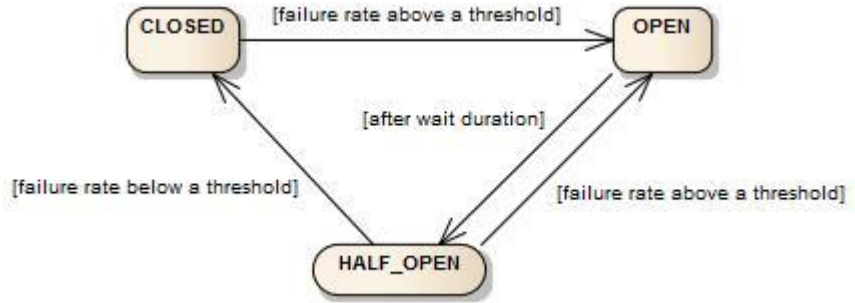


- F became unresponsive

Circuit breaker: the solution

“A service client should invoke a remote service via a proxy that functions in a similar fashion to an electrical circuit breaker.”

- If a call fails, increment the number of failed calls by one
- If the number of failed calls goes above a certain threshold, open the circuit
- If the circuit is open, immediately return with an error **or a default response**
- If the circuit is open and some time has passed, half-open the circuit
- If the circuit is half-open and the next call fails, open it again
- If the circuit is half-open and the next call succeeds, close it



Strategy	Implementations	Fit
Black Box	• Proxies • Service meshes	Fail fast
White Box	Libraries (e.g. Resilience4j)	Fallbacks relying on business logic

https://www.youtube.com/watch?v=kR2sm1zell4&ab_channel=GOTOConferences

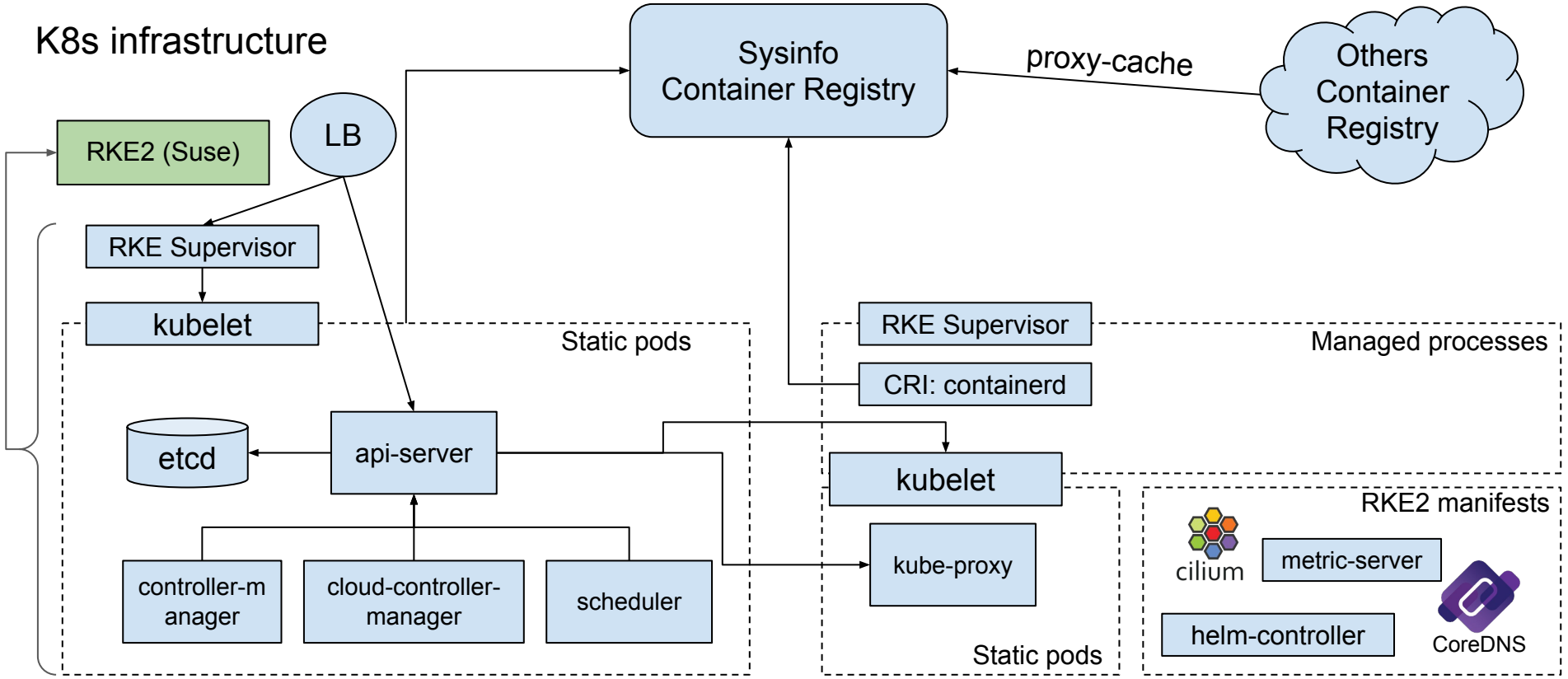
Kubernetes infrastructure high level components

1. Kubernetes (“k8s” - k+8 caratteri+s):
    a. *“ ...a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.”*
    b. Google open-sourced the Kubernetes project in 2014
    c. Features:
        - Service discovery, load balancing, horizontal scaling
        - Self-healing
        - Automated rollouts and rollbacks
        - Secret and configuration management

2. Falco (runtime security)

3. Container registry (Harbor)

K8s infrastructure

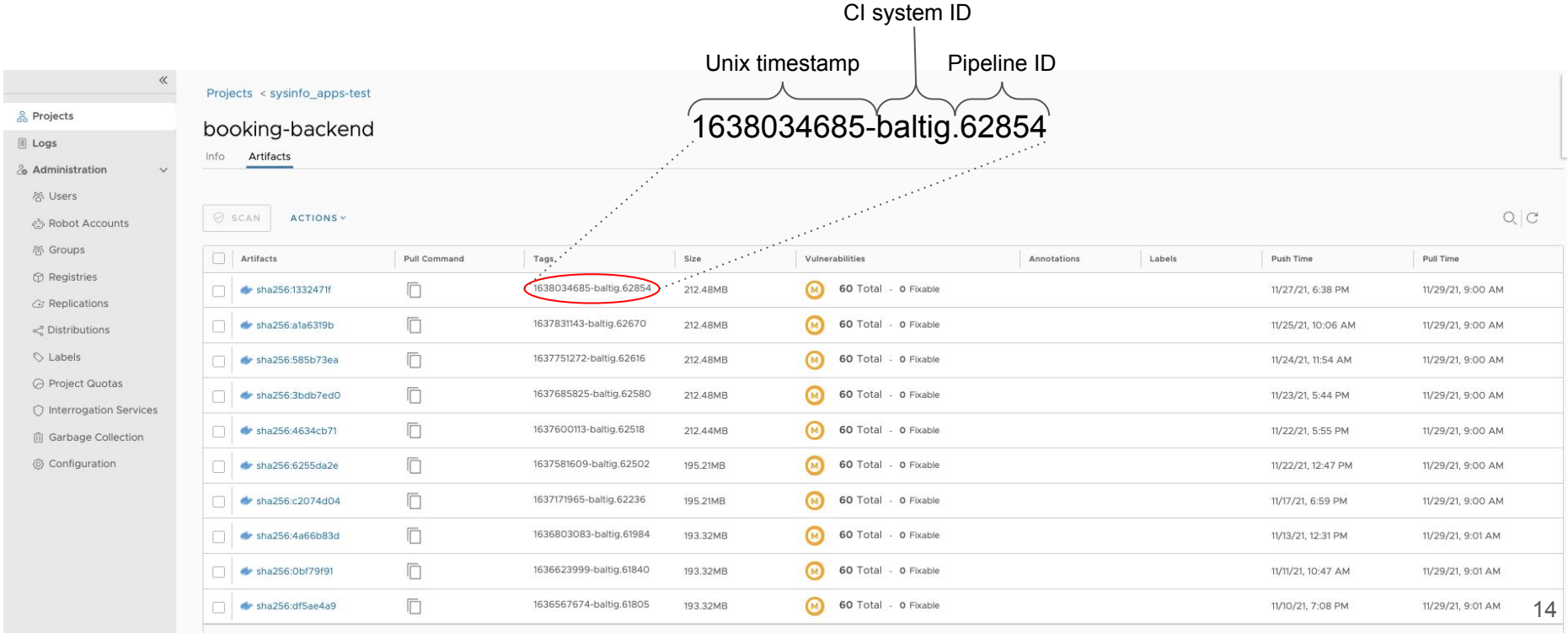


Base infra components (Puppet, Firewall, base monitoring etc.) + Falco (Kubernetes threat detection engine)

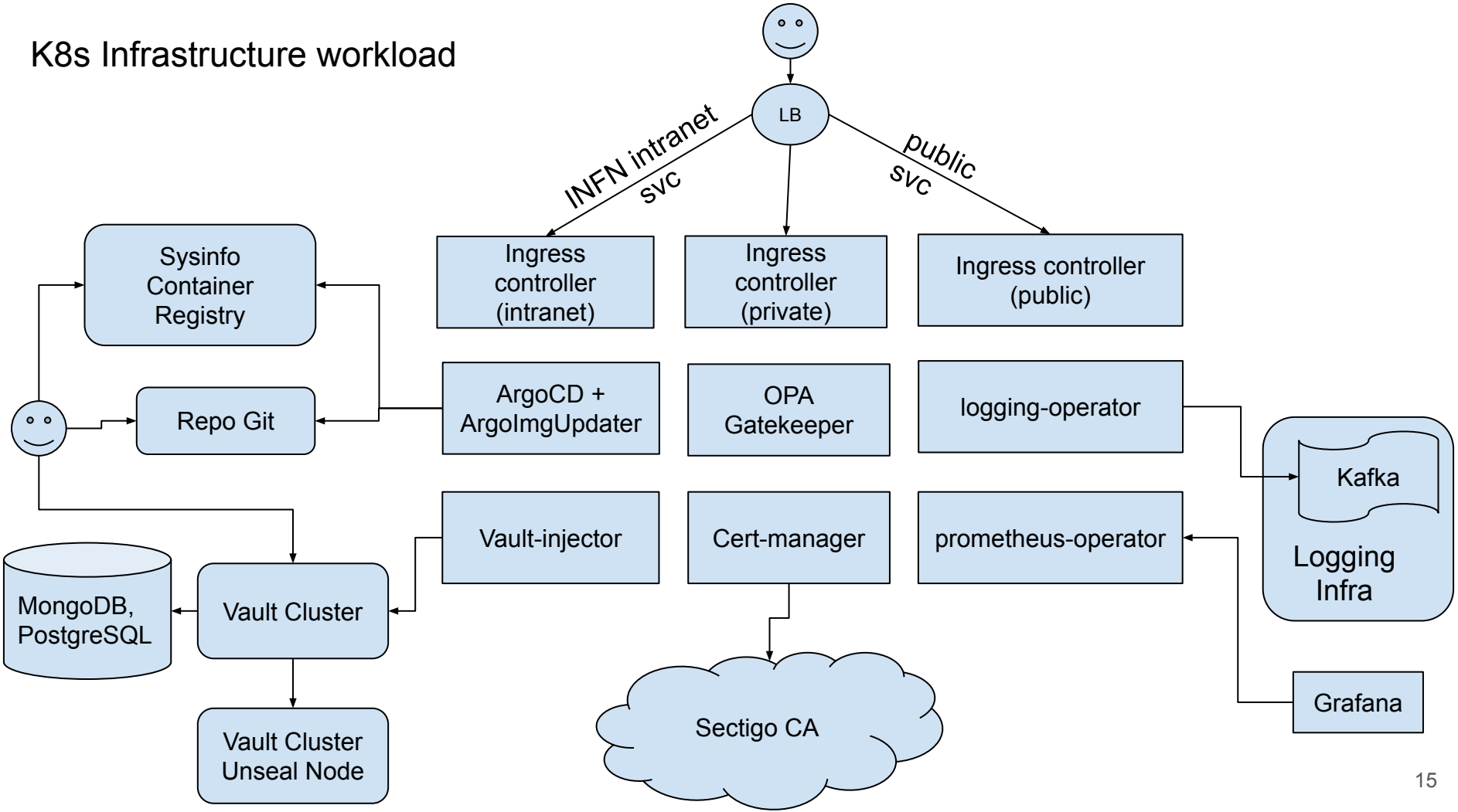
K8s Infra workload (some components must run on every node)

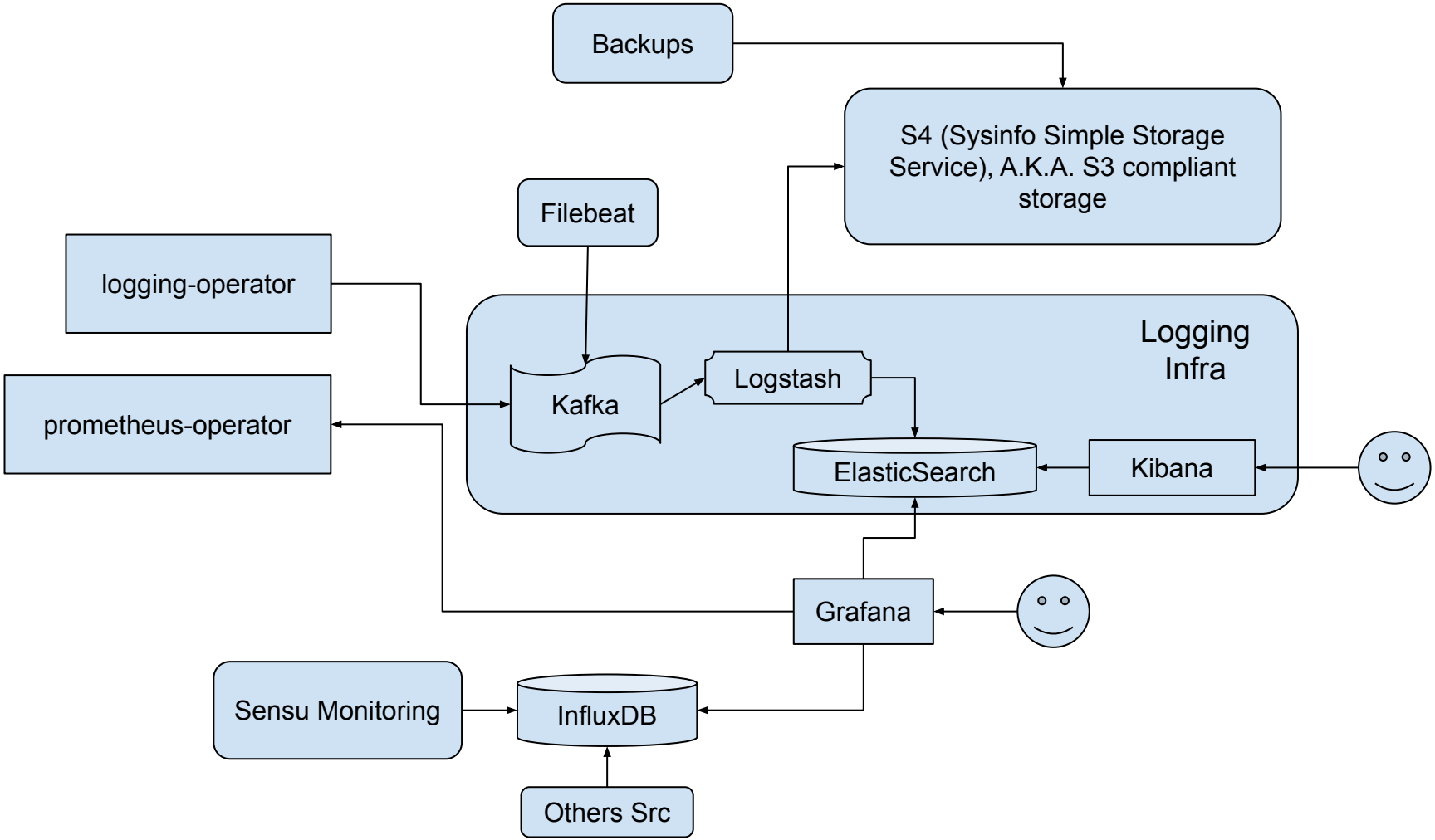
K8s Apps workload

Container registry



K8s Infrastructure workload

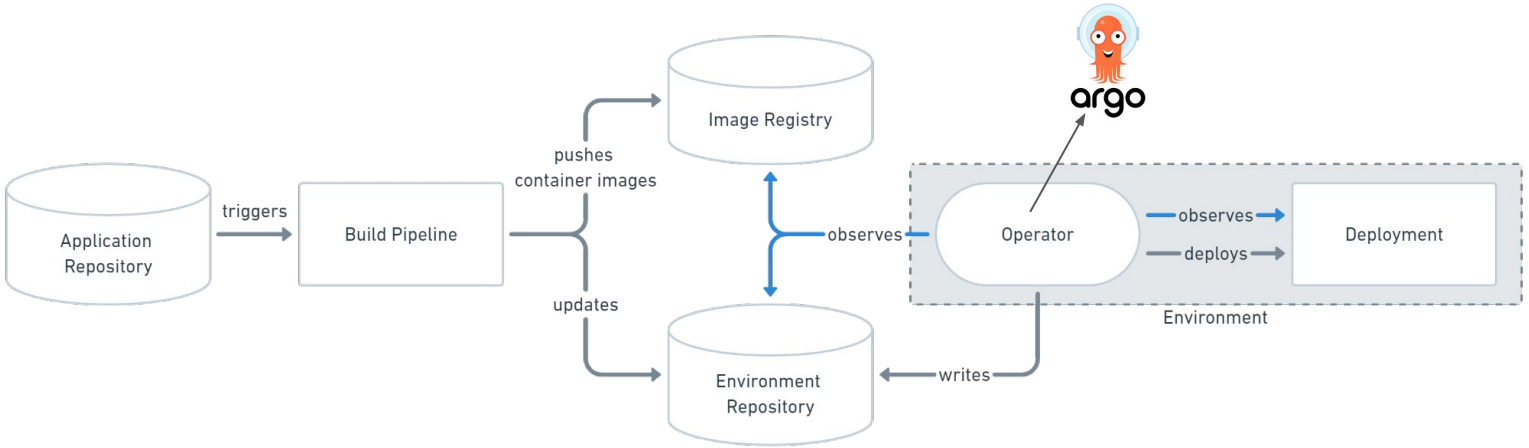



Backups
S4 (Sysinfo Simple Storage Service), A.K.A. S3 compliant storage
Filebeat
logging-operator
Logging Infra
Kafka
Logstash
prometheus-operator
ElasticSearch
Kibana
Grafana
Sensu Monitoring
InfluxDB
Others Src

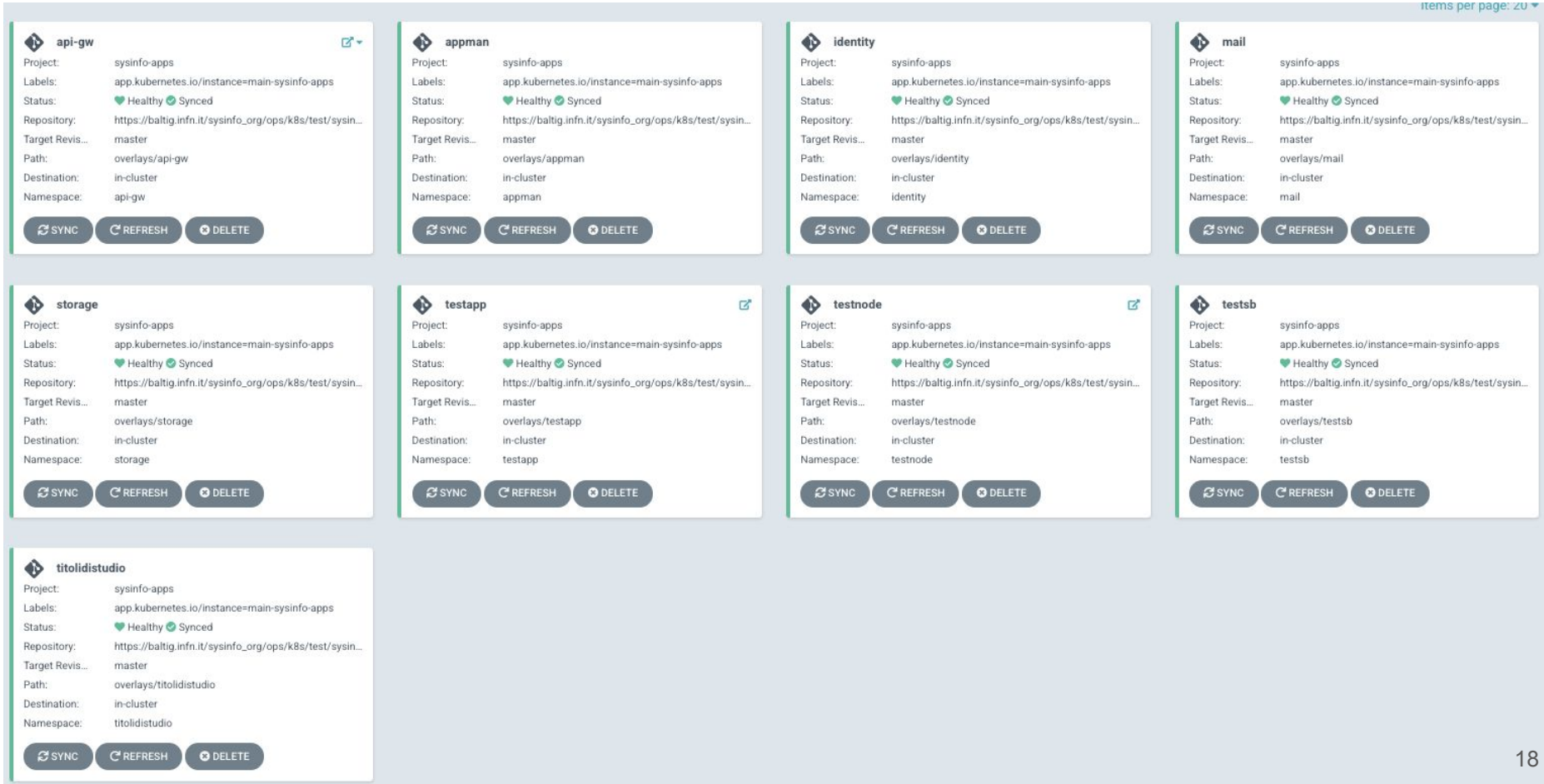
ArgoCD + GitOps

GitOps: versioned CI/CD on top of declarative infrastructure:

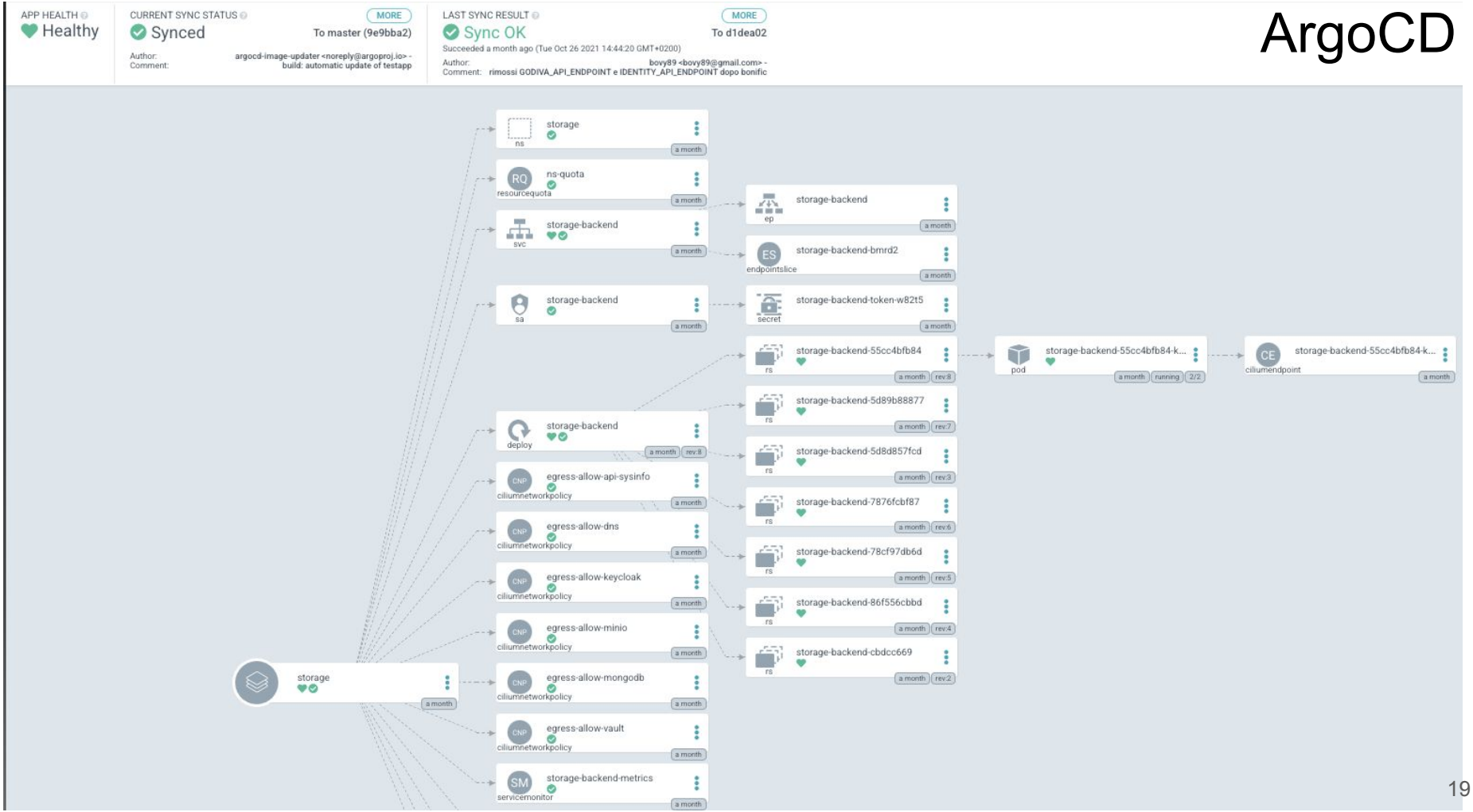
- a Git repository that always contains declarative descriptions of the desired infrastructure state
- infrastructure state versioned in Git
- use of continuous integration/continuous delivery pattern
- automated processes to make the production environment match the described state in the repository
- a change to the infrastructure (e.g. a new application) => modify the repository



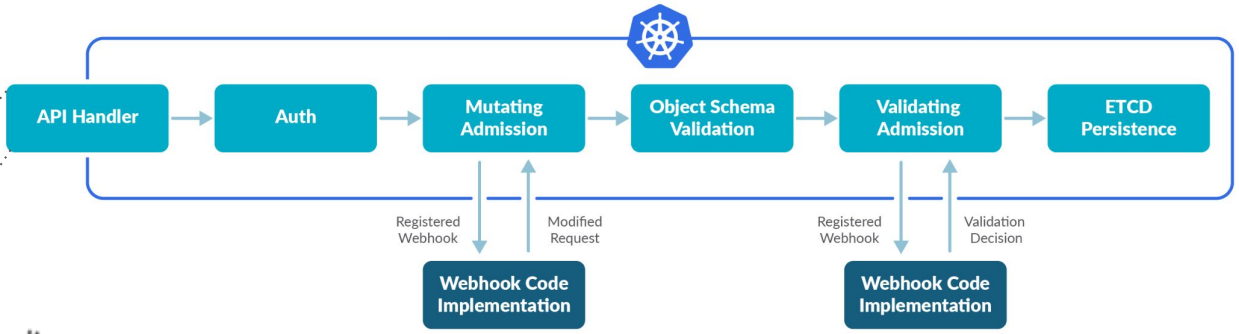
ArgoCD

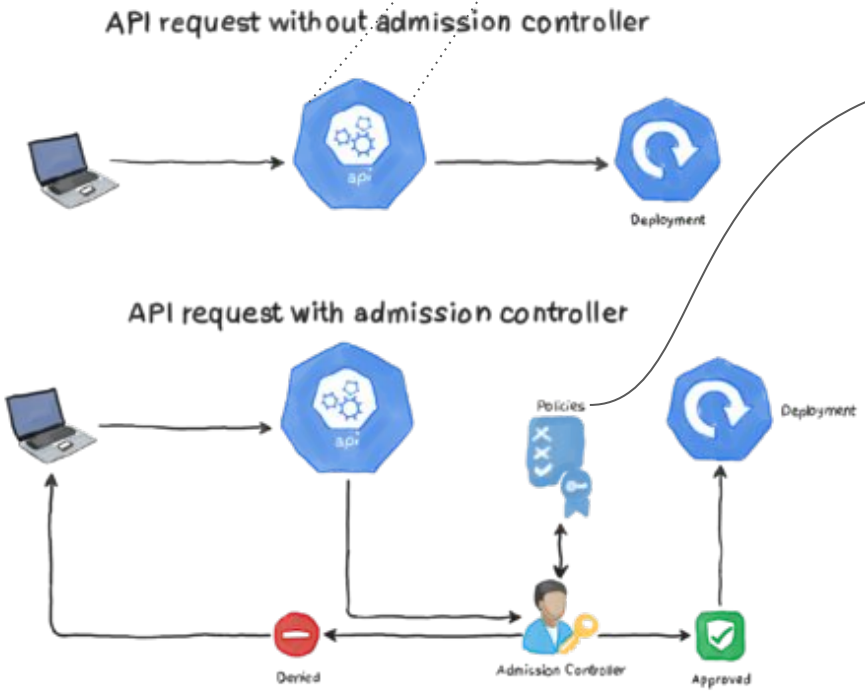

Items per page: 20
api-gw
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/api-gw
Destination: in-cluster
Namespace: api-gw
SYNC REFRESH DELETE
appman
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/appman
Destination: in-cluster
Namespace: appman
SYNC REFRESH DELETE
identity
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/identity
Destination: in-cluster
Namespace: identity
SYNC REFRESH DELETE
mail
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/mail
Destination: in-cluster
Namespace: mail
SYNC REFRESH DELETE
storage
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/storage
Destination: in-cluster
Namespace: storage
SYNC REFRESH DELETE
testapp
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/testapp
Destination: in-cluster
Namespace: testapp
SYNC REFRESH DELETE
testnode
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/testnode
Destination: in-cluster
Namespace: testnode
SYNC REFRESH DELETE
testsb
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/testsb
Destination: in-cluster
Namespace: testsb
SYNC REFRESH DELETE
titolidistudio
Project: sysinfo-apps
Labels: app.kubernetes.io/instance=main-sysinfo-apps
Status: Healthy Synced
Repository: https://baltig.infn.it/sysinfo_org/ops/k8s/test/sysin...
Target Revis... master
Path: overlays/titolidistudio
Destination: in-cluster
Namespace: titolidistudio
SYNC REFRESH DELETE

ArgoCD


APP HEALTH
Healthy
CURRENT SYNC STATUS
Synced
MORE
To master (9e9bba2)
Author: argocd-image-updater <noreply@argoproj.io> -
Comment: build: automatic update of testapp
LAST SYNC RESULT
Sync OK
MORE
To d1dea02
Succeeded a month ago (Tue Oct 26 2021 14:44:20 GMT+0200)
Author: bovy89 <bovy89@gmail.com> -
Comment: rimossi GODIVA_API_ENDPOINT e IDENTITY_API_ENDPOINT dopo bonific
storage
ns
a month
ns-quota
resourcequota
storage-backend
svc
storage-backend
ep
storage-backend-bmrd2
endpointslice
storage-backend
sa
storage-backend-token-w82t5
secret
storage-backend-55cc4bfb84
rs
storage-backend-55cc4bfb84-k...
pod
running
2/2
storage-backend-55cc4bfb84-k...
ciliumendpoint
storage-backend-5d89b88877
storage-backend
deploy
rev:8
storage-backend-5d8d857fcd
rev:7
rev:3
egress-allow-api-sysinfo
ciliumnetworkpolicy
storage-backend-7876fcbf87
rev:6
egress-allow-dns
storage-backend-78cf97db6d
rev:5
egress-allow-keycloak
storage-backend-86f556cbbd
rev:4
egress-allow-minio
storage-backend-cbdcc669
rev:2
storage
egress-allow-mongodb
egress-allow-vault
storage-backend-metrics
servicemonitor

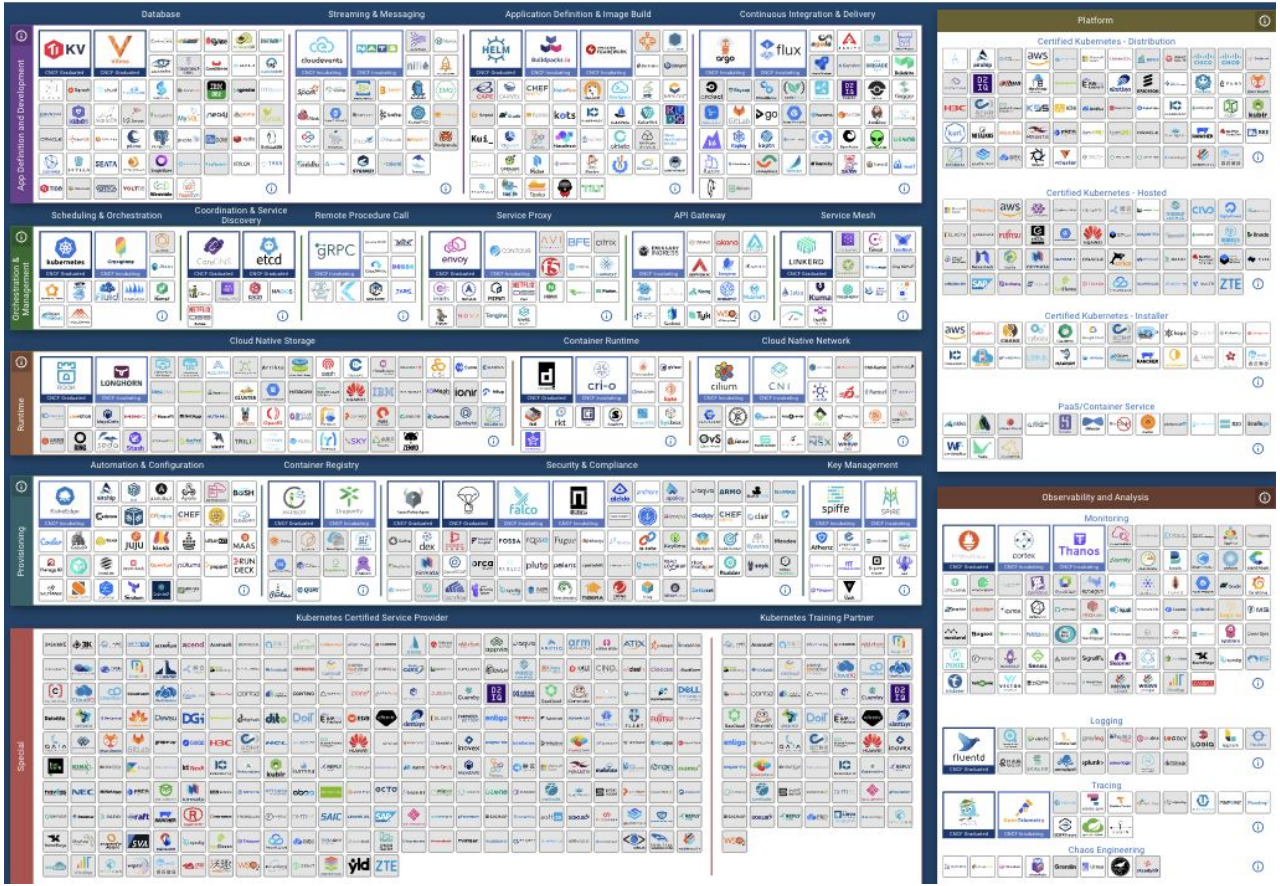
OPA Gatekeeper





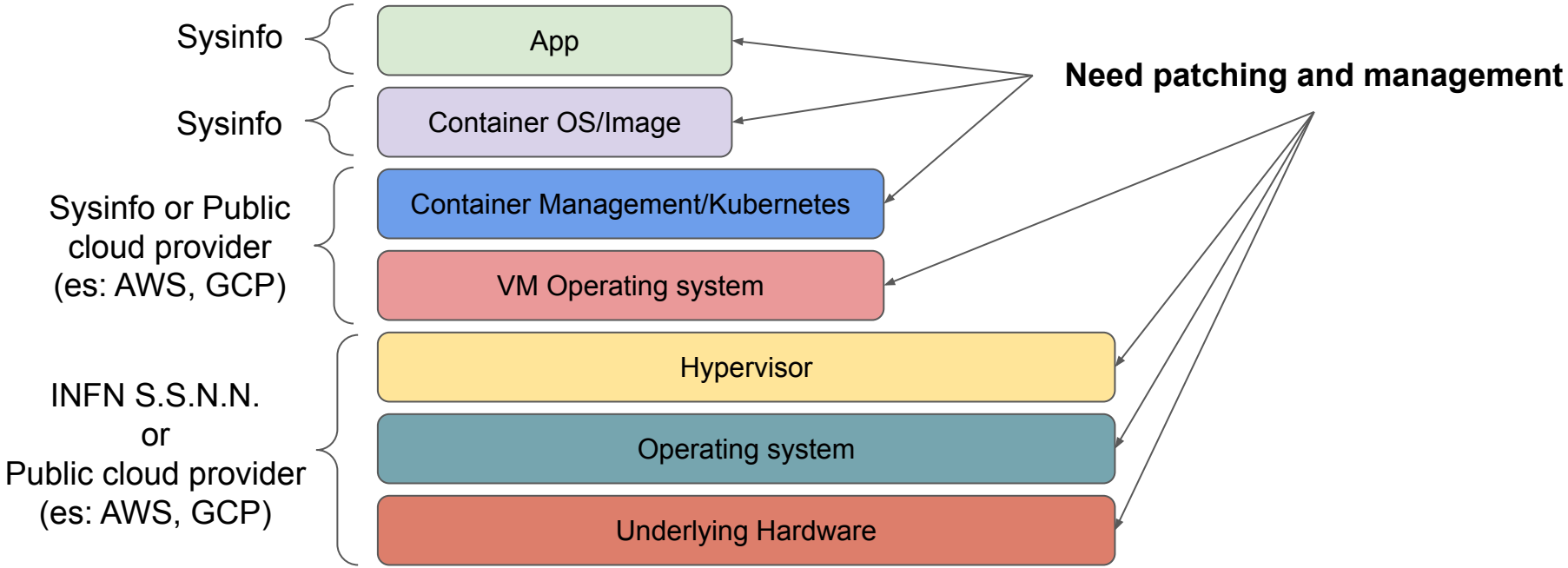

```
package k8spspallowprivilegeescalationcontainer

violation[{"msg": msg, "details": {}}] {
    c := input_containers[_]
    input_allow_privilege_escalation(c)
    msg := sprintf("Privilege escalation container is not allowed: %v", [c.name])
}

input_allow_privilege_escalation(c) {
    not has_field(c, "securityContext")
}
input_allow_privilege_escalation(c) {
    not c.securityContext.allowPrivilegeEscalation == false
}
input_containers[c] {
    c := input.review.object.spec.containers[_]
}
input_containers[c] {
    c := input.review.object.spec.initContainers[_]
}
# has_field returns whether an object has a field
has_field(object, field) = true {
    object[field]
}
```

How do we secure this?




I'M SCANNING
MY CONTAINER IMAGES

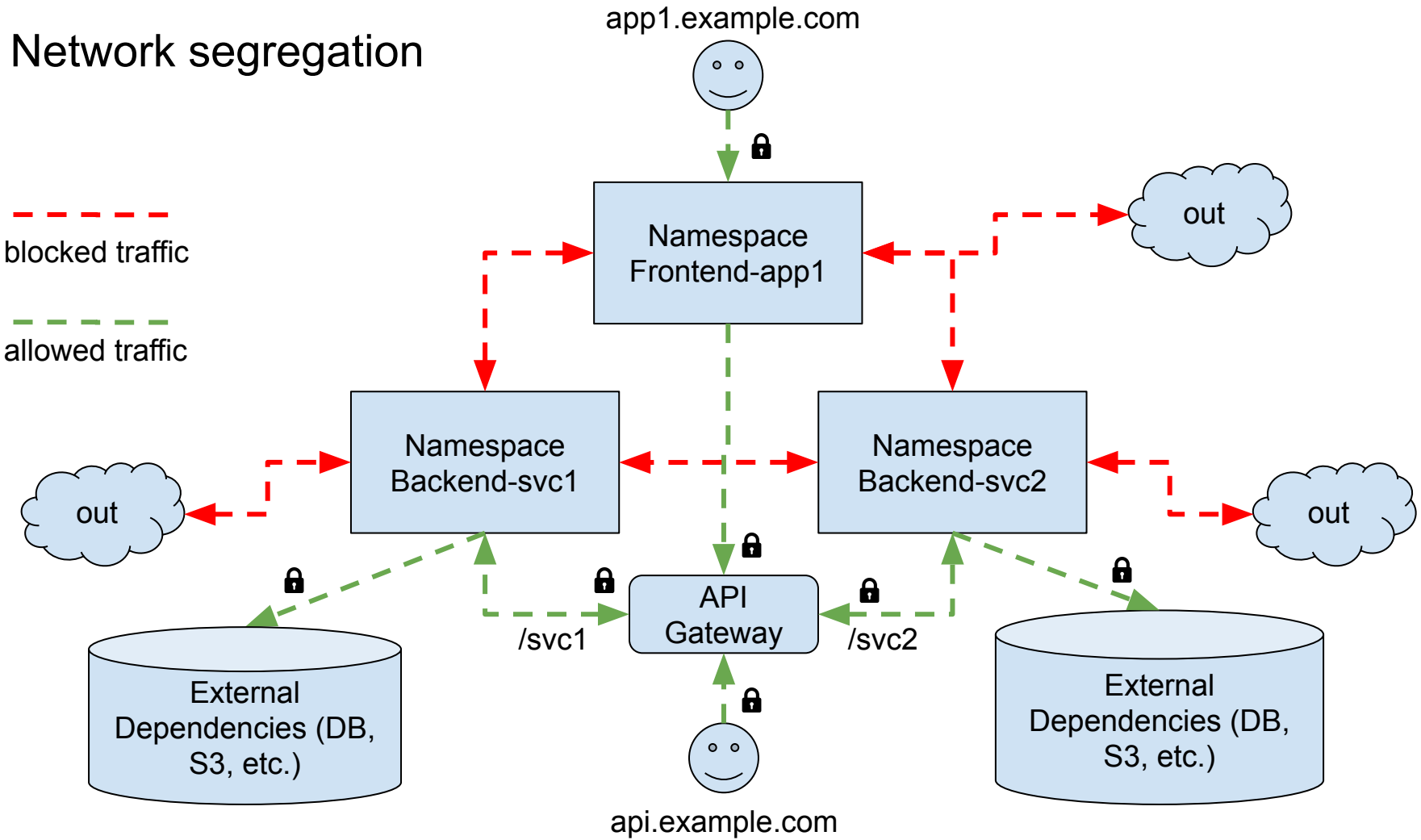
Infrastructure patching&management



Infrastructure security management

- GitOps approach
- Static analysis + Deprecations checks
- Unit testing
- Rego policy to enforce K8s best practices and security issues mitigation/prevention:
  - CI job ---> using Trivy scanner
  - K8s API webhook ---> using OPA Gatekeeper
- Vulnerability scanning
- Minimal base images
- Container image scan before push/after push/periodically (every day)
- Ossec + Falco + Observability via Cilium
- Network Policy
- Seccomp / AppArmor
- Log analisys&monitoring

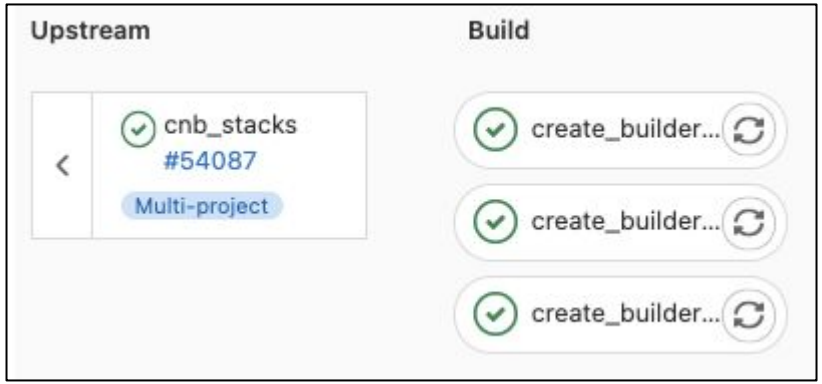
Network segregation



Stack Images (Scheduled pipeline every 24h)

1. Build stack Image
2. Security scan on builded image
3. Push builded image into a temporary area (test)
4. Test pushed image (e.g. test java build and execution)
5. Promote image from temporary area (test) to prod (will be used to build our projects)
6. Trigger builder images rebuild

Builder Images (Triggered every 24h by build images pipeline)



1. Build Image
2. Test builded image (e.g. execute and check as springboot application)
3. Security scan on builded image
4. Push builded image to prod (will be used to run projects)

Novità lato CI/CD

- I Tag non sono necessari per le pipeline relative al workflow “container”
- Pipeline e flusso di lavoro molto più user-friendly e lineare
- Rimosso utilizzo di long lived branches (vedi development):
  - utilizzo effettivo della CI
  - utilizzo di approcci Dark launch, Feature toggles ecc
- Tutti i job sono in modalità “strict”: failure => blocco pipeline
- Test job (unit test): al momento solo per backend (vedi slide successive)
- Unit test (TDD sarebbe una buona prassi): richiesto coverage > 90% (*)
- Accessibility (A11Y) job (Frontend only): unico job in “tech preview” in modalità non strict
- Aggiunta generazione e analisi SBOM (Software bill of materials) via Dtrack
- Deployment via container images e non via RPM

(*) vi invito caldamente a leggere questo articolo per capirne l’importanza che va ben oltre la percentuale indicata

Backend (Java) pipeline

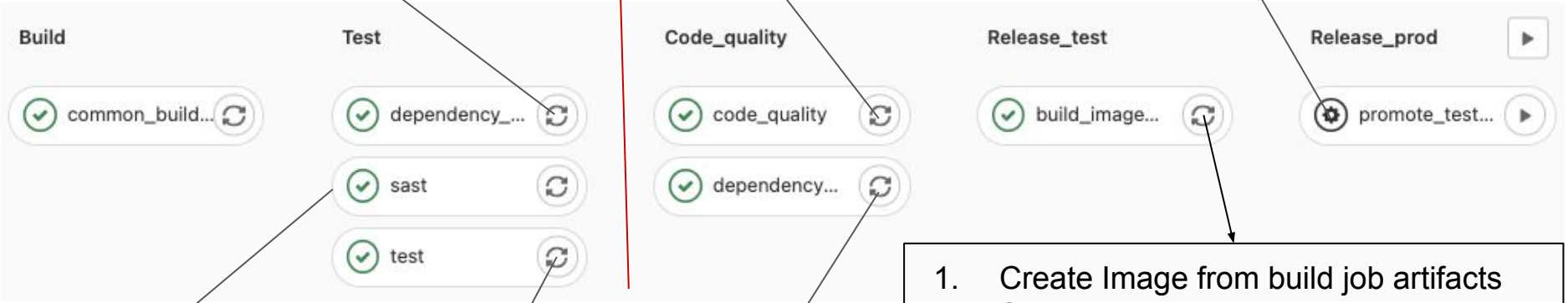
if branch != DEFAULT_BRANCH stop here

Owasp dependency check

Sonarqube

NOTE: Manually triggered

1. Security scan on test (src) image
2. Promote src image from test to prod
3. Pushed image will be automatically deployed by Argocd (prod environment)



Static Analysis (security only)

Unit Test

Dtrack

1. Create Image from build job artifacts
2. Security scan on builded image
3. Push builded image to test environment
4. Pushed image will be automatically deployed by Argocd (test environment)

Frontend (NodeJS) pipeline

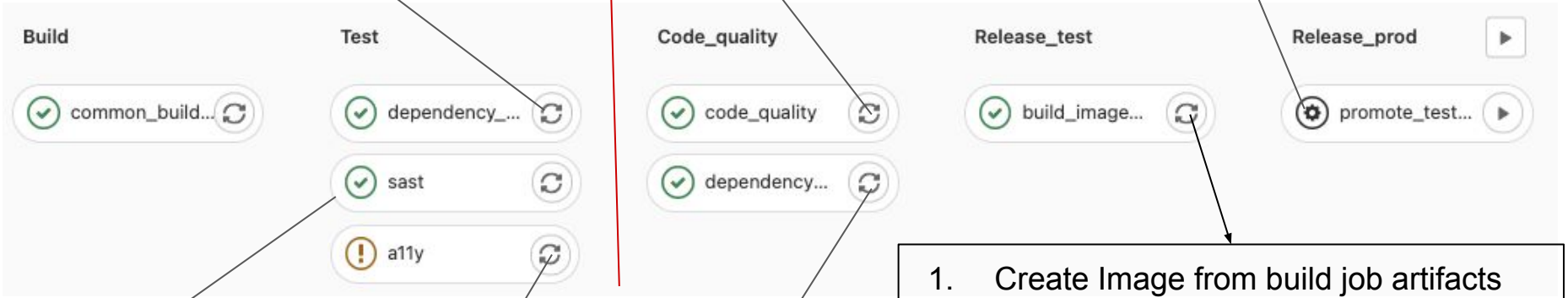
if branch != DEFAULT_BRANCH stop here

Owasp dependency check

Sonarqube

NOTE: Manually triggered

1. Security scan on test (src) image
2. Promote src image from test to prod
3. Pushed image will be automatically deployed by Argocd (prod environment)



Linter

A11Y Test

Dtrack

1. Create Image from build job artifacts
2. Security scan on builded image
3. Push builded image to test environment
4. Pushed image will be automatically deployed by Argocd (test environment)

Test job (backend only)

- Sfrutta docker-compose
- Prevede servizi docker-compose sulla falsariga di quelli in produzione (es: MongoDB, Minio ecc)
- Servizi “mocked” dinamici sulla base delle esigenze del progetto
- Servizio docker-compose “test”: esegue effettivamente i test
- Lo sviluppatore può eseguire lo stesso job della CI in locale
- Lo sviluppatore configura in totale autonomia la configurazione necessaria per i test sulla base delle dipendenze dell’applicazione, esempio:

```
{
  "mongodb": {},
  "postgres": {},
  "mail": {
    "mock_path": "test-rsrc/mocks/mail.json",
    "port": 8082
  }
}
```

Accessibility job (frontend only)

- Utilizza PA11Y (lo stesso usato da Gitlab)
- Scansione mediante engine Axe e HTML Code Sniffer
- Scansione proattiva (prima che venga rilasciato in test/prod)
- Lo sviluppatore può eseguire gli stessi test in locale
- Identifica i problemi di base senza setup complicati o configurazioni da parte dello sviluppatore

NOTE:

- Raffinamenti del job ancora in corso
- Job in modalità non strict in attesa di verifiche/considerazioni sui progetti software
- Svariati problemi già identificati: test automatizzati di questo tipo non troveranno mai tutti i problemi
- Per identificare e sistemare tutti i problemi è comunque consigliato utilizzare i plugin del browser

App security patch management

Le patch necessarie a livello applicativo sono molto più frequenti di quelle lato OS: più impegno richiesto da parte vostra

La priorità numero 1 deve essere tenere il software funzionante, aggiornato e sicuro (non trascurate Sonarqube)

I test per identificare problemi di security:

- devono essere fatti in locale durante lo sviluppo software
- vengono eseguiti in automatico dai job di CI

Nei job di CI i security scan sono bloccanti se vulnerabilità >= high: controllare sempre i report generati e controllare Dtrack (anche se le vulnerabilità sono low/medium è bene aggiornare)

Se esce una vulnerabilità >= high bisogna aggiornare ASAP (non dopo settimane/mesi)

Se esce una vulnerabilità < high bisogna comunque aggiornare ma possiamo farlo con più calma (settimane)

I framework/librerie devono sempre essere con supporto attivo (NON in EOL)

Va considerato almeno quanto sopra nel calcolo dell'effort richiesto per mantenere/scrivere una "nuova applicazione"

Il branch principale deve poter essere rilasciato in prod in un qualunque momento

Ma non sarà un pò troppo?

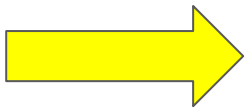
“Without requirements or design, programming is the art of adding bugs to an empty text file.”

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.”

High quality software is cheaper to produce:

- Neglecting internal quality leads to rapid build up of cruft[1]
- This cruft slows down feature development
- Even a great team produces cruft, but by keeping internal quality high, is able to keep it under control
- High internal quality keeps cruft to a minimum, allowing a team to add features with less effort, time, and cost.



https://martinfowler.com/articles/is-quality-worth-cost.html

1) cruft: badly designed, unnecessarily complicated, or unwanted code or software.

E i database?

Il database di riferimento è MongoDB. Se è necessario un DB relazionale, la engine di riferimento è PostgreSQL.

A tendere, a seguito di processo di bonifica/migrazione ad un approccio a microservizi, il dato sarà spostato da Oracle verso altre DB Engine (vedi sopra).

Lo sviluppo software non deve prevedere l'utilizzo di database remoti: tutto locale con dati "finti".

Gli account sui DB per il personale "interno" sono previsti esclusivamente in sola lettura e saranno creati on-demand (durata max nell'ordine di qualche ora).

Il data correction "fisiologico" non è previsto: il software deve funzionare. In caso di malfunzionamenti, il problema va sistemato a monte e va prodotta una patch per la bonifica dei dati.

L'accesso al DB sarà una attività saltuaria per identificare/riprodurre problemi e patch.

Le modifiche alla struttura dati/oggetti del DB devono fare parte del software stesso:

- versionate
- gestite mediante appositi framework
- applicate automaticamente mediante script di "migrazione"

NOTA: dovremo sempre lavorare con retrocompatibilità almeno N-1

E i file/documenti generati dalle applicazioni?

- La maggior parte dei documenti è al momento su filesystem
- Alcune applicazioni utilizzano Alfresco
- C’è bisogno di uno strumento/set di API standard (ampiamente supportate) e adatto al tipo di infrastruttura/applicazioni che stiamo progettando: S3 (Minio)
- E’ già previsto un microservice “storage” che utilizza Minio per la gestione dei documenti (per dettagli vedi prossime presentazioni)
- Il “ciclo di vita” dei documenti generati dalle applicazioni va sicuramente ridiscusso
- Il “ciclo di vita” a fini normativi dei documenti generati dalle applicazioni è tutto da capire/definire e da progettare/implementare (a prescindere dallo storage) in quanto al momento non è comunque presente nulla di strutturato e automatico

Stato dell’arte

- Infrastruttura predisposta per test e produzione
- Casi d’uso principali pronti (Springboot e NodeJS)
- Le applicazioni “pronte” sono state rilasciate in test
- Servizi ancillari:
    - Cache Redis su K8s da finalizzare + da collaudare interazione con backend
    - Kafka: da collaudare interazione con backend
    - PostgreSQL da finalizzare
- Taks da terminare prima della messa in prod delle applicazioni:
    - Rifiniture su applicazioni “in test”
    - Rifiniture monitoraggio e logging
    - Security scan e audit (parzialmente fatti)
    - Implementare/migliorare procedure automatizzate di validazione del cluster (per velocizzarne upgrade futuri)
    - Da schedulare minor upgrade k8s e applicazioni “infrastrutturali”

Possiamo partire a razzo e deployare nuove applicazioni? **No** ---> vedi prossima slide

Punti aperti

1. Dobbiamo definire bene gli obiettivi e il path di migrazione Monolith--->Microservices (non ancora definito/discusso/analizzato): non possiamo navigare a vista. **NOTA: migrare ad una architettura a microservizi NON è l'obiettivo**
2. Una modifica alla volta (No big-bang rewrite o troppi cambiamenti drastici che portano ad una sorta di lock-in)
3. Necessaria formazione del personale: nuovi framework, nuovi linguaggi, nuovo modo di lavorare, molti aspetti tecnici (e non) da considerare
4. Dopo la formazione, serve affiancamento/review con "gli esperti" e tempo per prendere confidenza
5. Lato infrastruttura, oltre ai task programmati, saranno sicuramente da rivalutare aggiunte/migliorie delle componenti core (prima che il sistema abbia 23432432 servizi):
    a. API gateway per altri requirements/casi d'uso evoluti
    b. service mesh
    c. distributed tracing e modifiche per correlation ID (al momento non previsto da logiche di parsing/configurazioni vare): deve essere fatto in parte a livello infrastrutturale e poi propagato a livello software
6. Review/eventuale path di migrazione per "Universo BI" ed infrastruttura sottostante ancora tutto da analizzare/discutere nel dettaglio

Risorse utili

https://samnewman.io/about/#work-with-me

https://martinfowler.com/

https://microservices.io/

Qui trovate i pattern per la migrazione sia delle applicazioni che dei dati