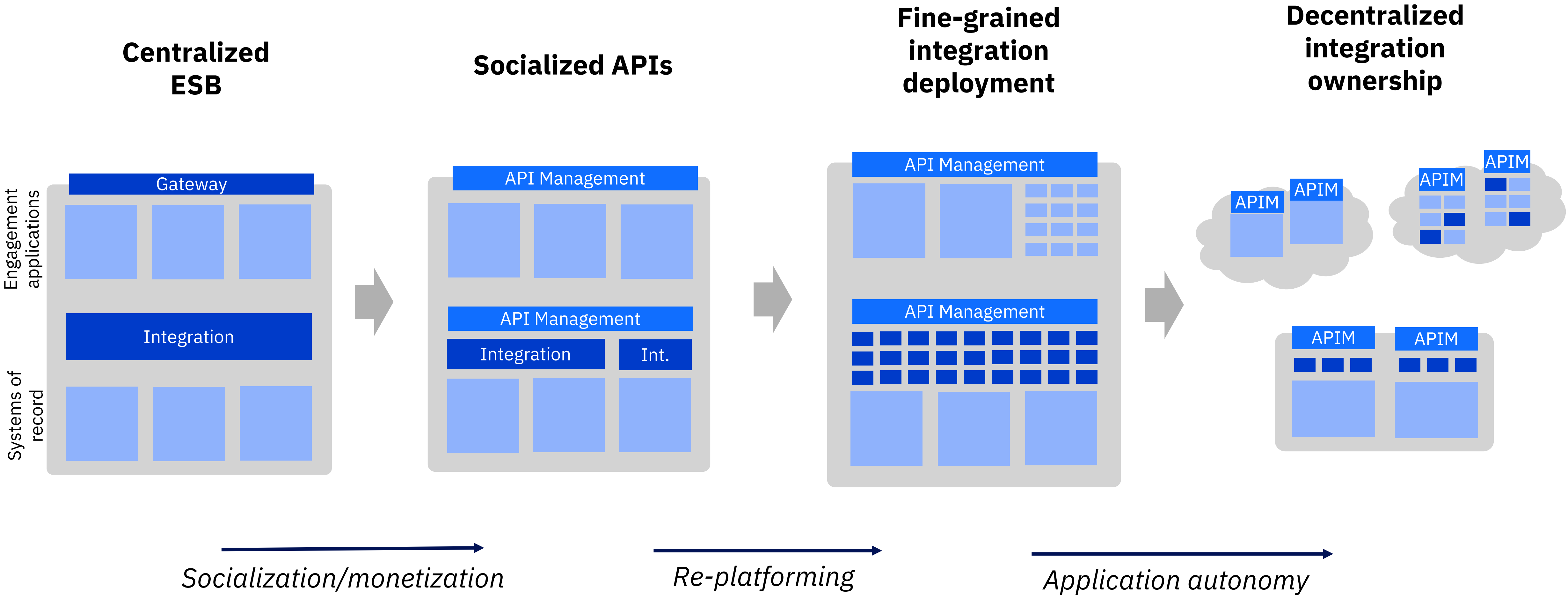
Cloud native for agile integration

Murali Sitaraman
Geo-Lead Integration
IBM Automation, EMEA
msit@ch.ibm.com

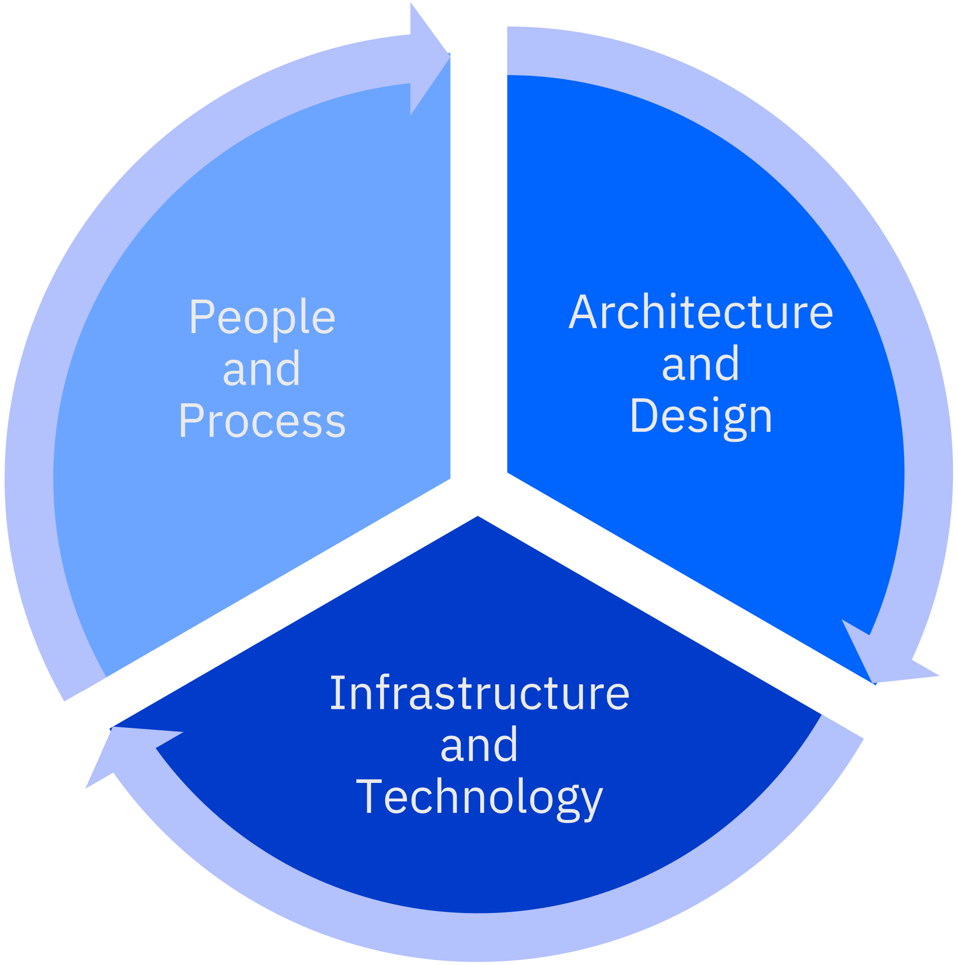
IBM Labs Come To You
26. April
Cologne

Evolution to **agile integration**



Webinar http://ibm.biz/agile-integration-webinar eBooklet http://ibm.biz/agile-integration IBM Redbook http://ibm.biz/agile-integration-redbook “cloud native” means

fully leveraging the uniqueness of cloud



"Cloud native" means ***fully leveraging the uniqueness of cloud*** *to achieve…*

Agility and Productivity

- Enable rapid innovation that is guided by business metrics.
- De-risk changes and maintenance and keep environments current.

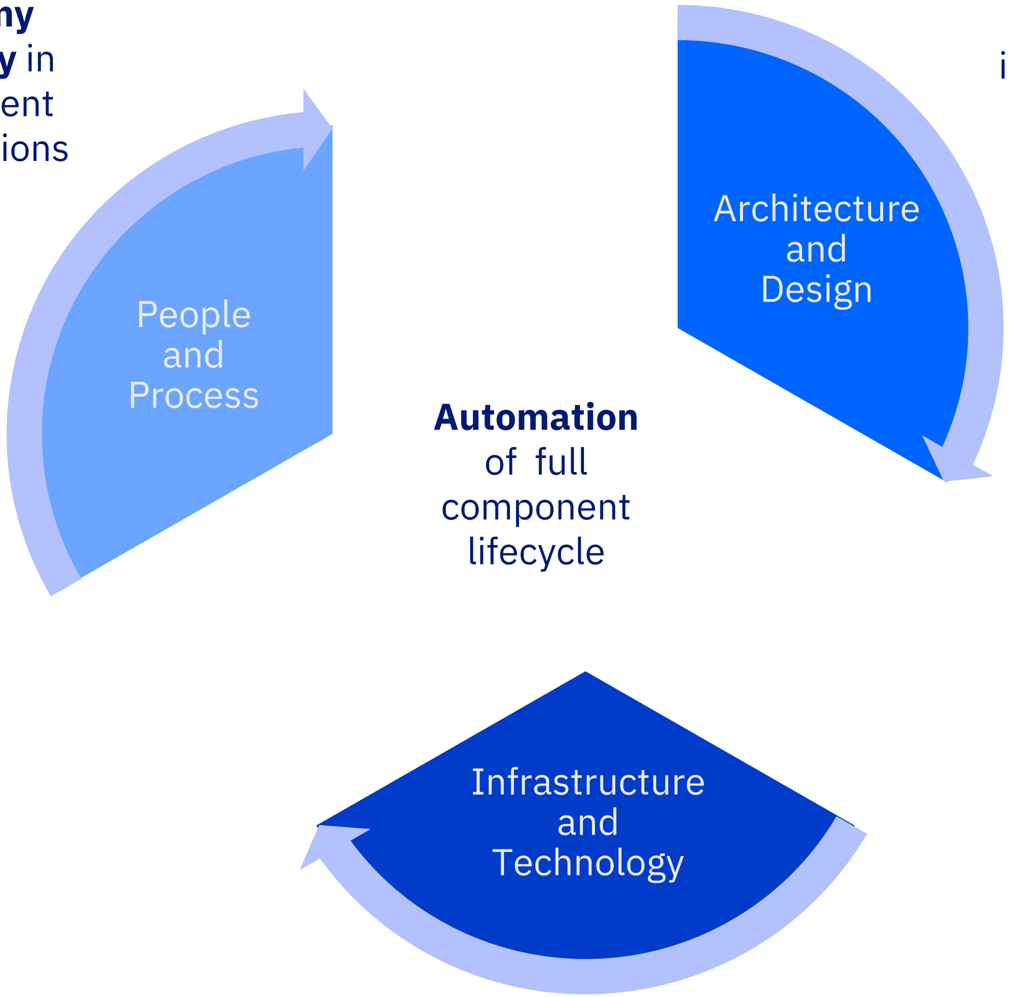
Resilience and Scalability

- Target continuous availability that is self-healing and downtime-free.
- Provide elastic scaling and the perception of limitless capacity.

Optimization and Efficiency

- Optimize the costs of infrastructural and human resources.
- Enable free movement between locations and providers.

…through…



Autonomy and agility in development and operations

Solutions that leverage infrastructure abstractions

Automation of full component lifecycle

Platforms that abstract complexities of infrastructure

http://ibm.biz/cloudnativedefined

Ingredients of cloud native

People and process

- Agile methods
- Lifecycle automation
- DevOps and SRE
- Team autonomy

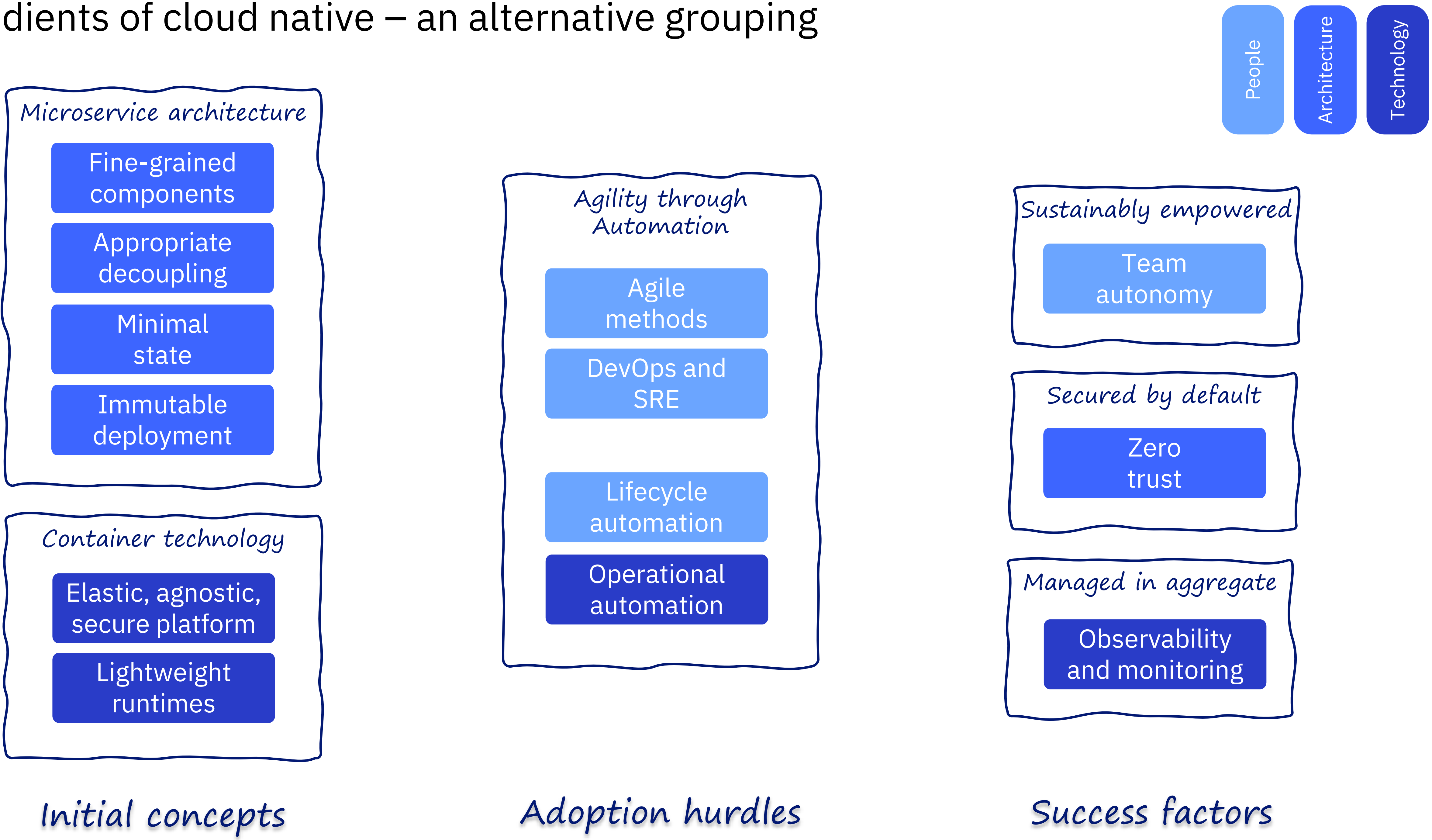
Architecture and design

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment
- Zero trust

Technology and infrastructure

- Elastic, agnostic, secure platform
- Lightweight runtimes
- Operational automation
- Observability and monitoring

Ingredients of cloud native – an alternative grouping



People

Architecture

Technology

Microservice architecture

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology

- Elastic, agnostic, secure platform
- Lightweight runtimes

Initial concepts

Agility through Automation

- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Adoption hurdles

Sustainably empowered

- Team autonomy

Secured by default

- Zero trust

Managed in aggregate

- Observability and monitoring

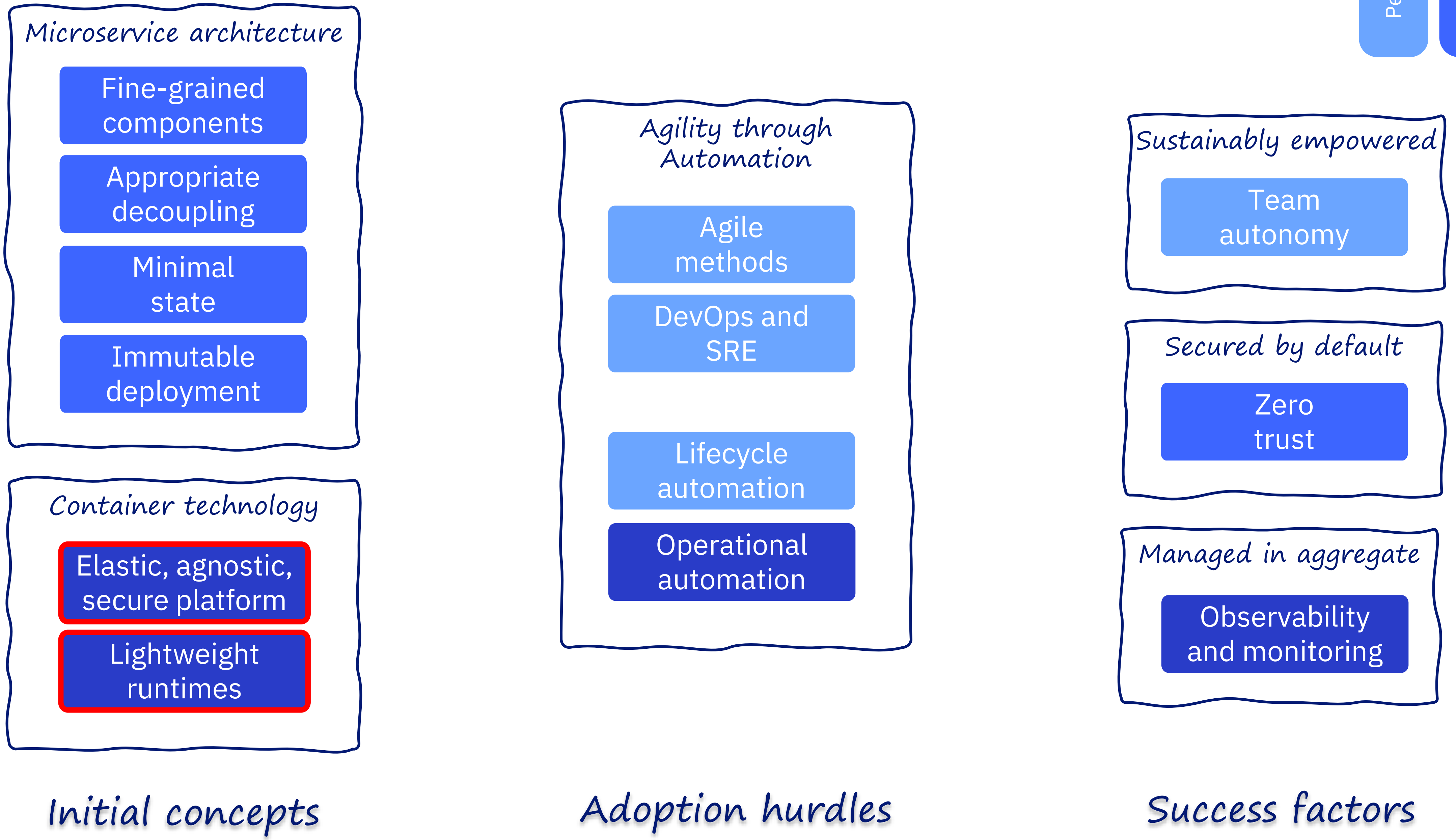
Success factors

Ingredients of cloud native – an alternative grouping

People

Architecture

Technology



Initial concepts

Microservice architecture

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology

- Elastic, agnostic, secure platform
- Lightweight runtimes

Adoption hurdles

Agility through Automation

- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Success factors

Sustainably empowered

- Team autonomy

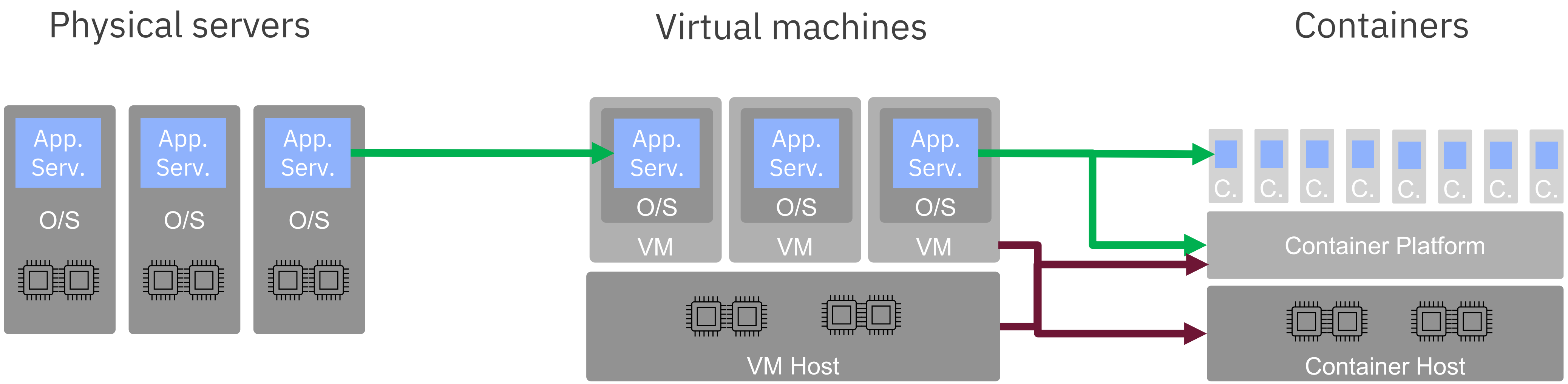
Secured by default

- Zero trust

Managed in aggregate

- Observability and monitoring

The move to containers is *very* different from the preceding move to virtual machines



Lift and shift to

- Optimize hardware
- Simplify provisioning

Refactor to realign responsibilities of application, app server and operating system, networking and storage to facilitate

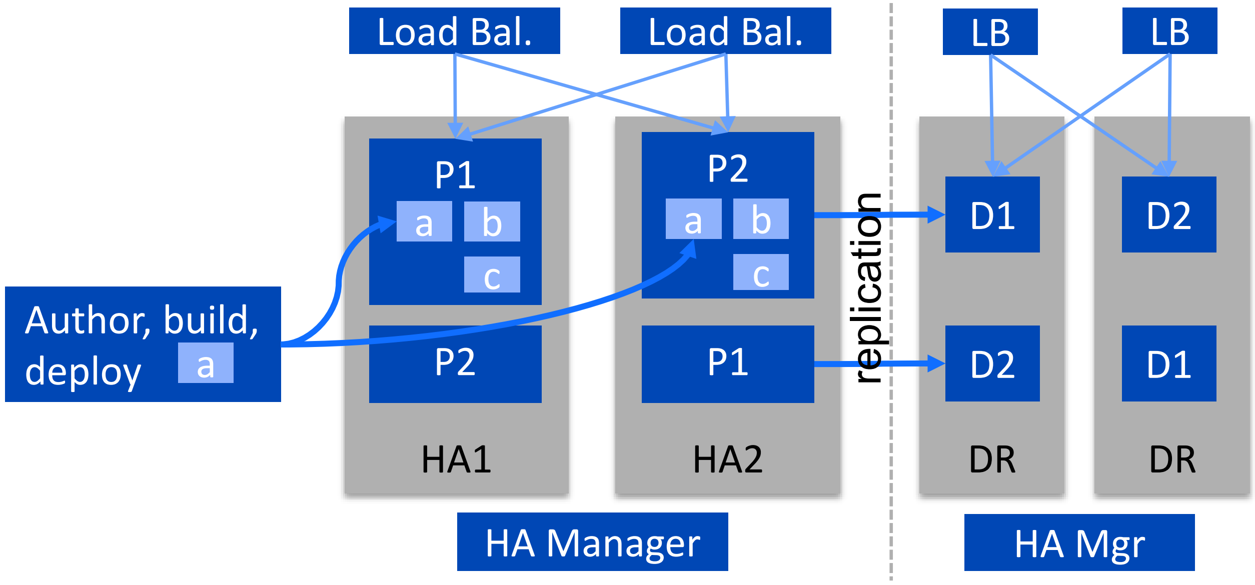
- Lifecycle agility through automation
- Rationalized operations across all runtime types
- Discrete, agnostic resilience and scalability

Traditional vs Cloud native

■ Product component ■ Product artefact

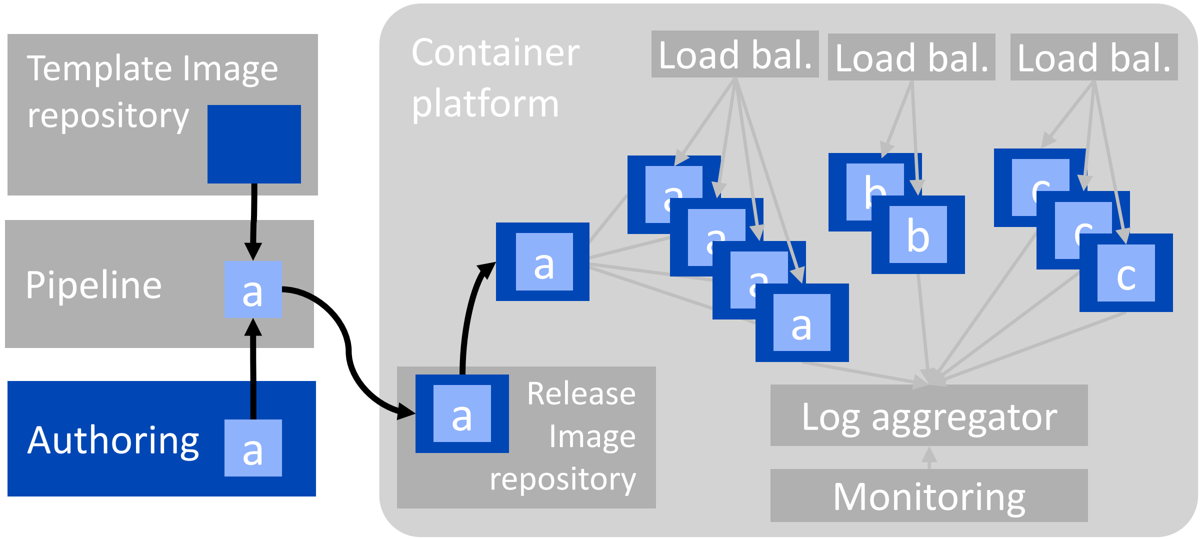
Traditional

- Dedicated HA pairs
- Scaling manual and vertical
- Defined nodes
- Explicit install and configure
- Explicit cold/warm HA & DR
- Dedicated OS instances/HW
- Deploy to running shared servers
- Replication across DCs
- Administer live shared servers
- Code deployed to shared servers



Cloud-native

- Elastically scaled containers
- Pooled underlying resources,
- Decoupled, fine-grained containers
- Implicit HA/DR
- Image based install and deployment
- Deployed and updated declaratively
- Administer by declarative infrastructure as code

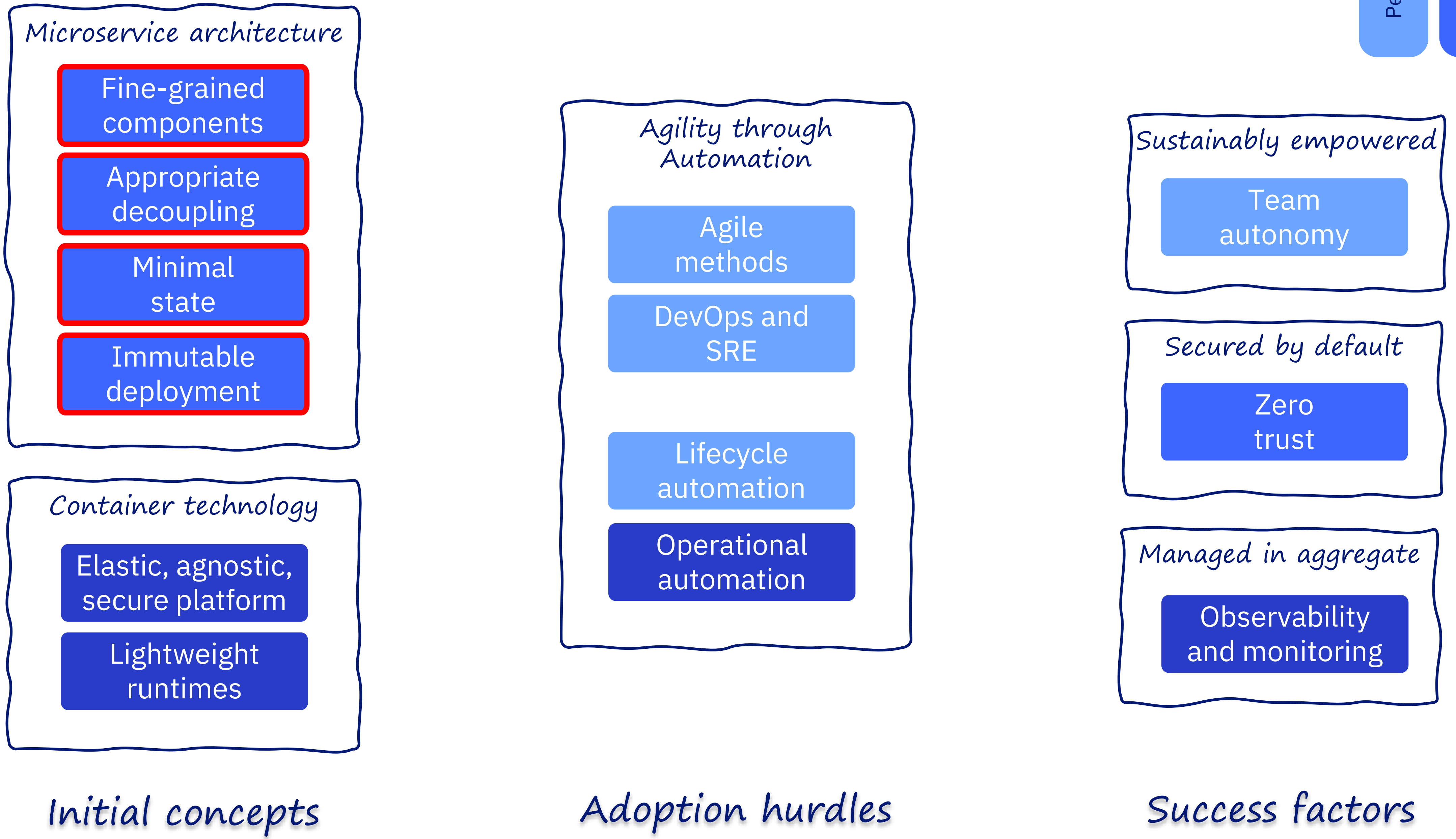


Ingredients of cloud native – an alternative grouping

People

Architecture

Technology



Microservice architecture

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology

- Elastic, agnostic, secure platform
- Lightweight runtimes

Initial concepts

Agility through Automation

- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Adoption hurdles

Sustainably empowered

- Team autonomy

Secured by default

- Zero trust

Managed in aggregate

- Observability and monitoring

Success factors

Microservice architecture – key concepts

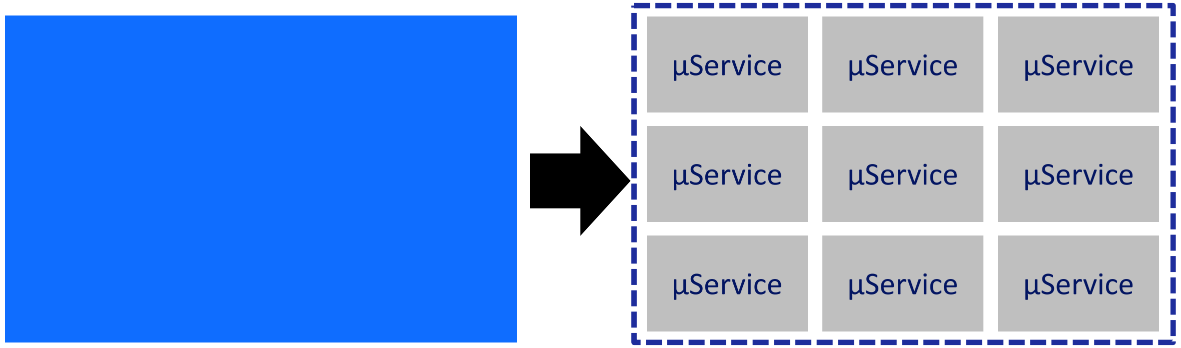
Microservice architecture

Fine-grained components
- Function driven granularity
- Self-contained components
- Independent lifecycles, scaling and resilience

Appropriate decoupling
- Clear ownership boundaries
- Formalised interfaces (API and event)
- Independent persistence

Minimal state
- Uncomplicated horizontal scaling
- No caller or session affinity
- No two phase commits

Immutable deployment
- Image based deployment
- No runtime administration
- Updates and rollbacks by replacement



Agility

Faster iteration cycles,
bounded contexts,
autonomous teams

Scalability

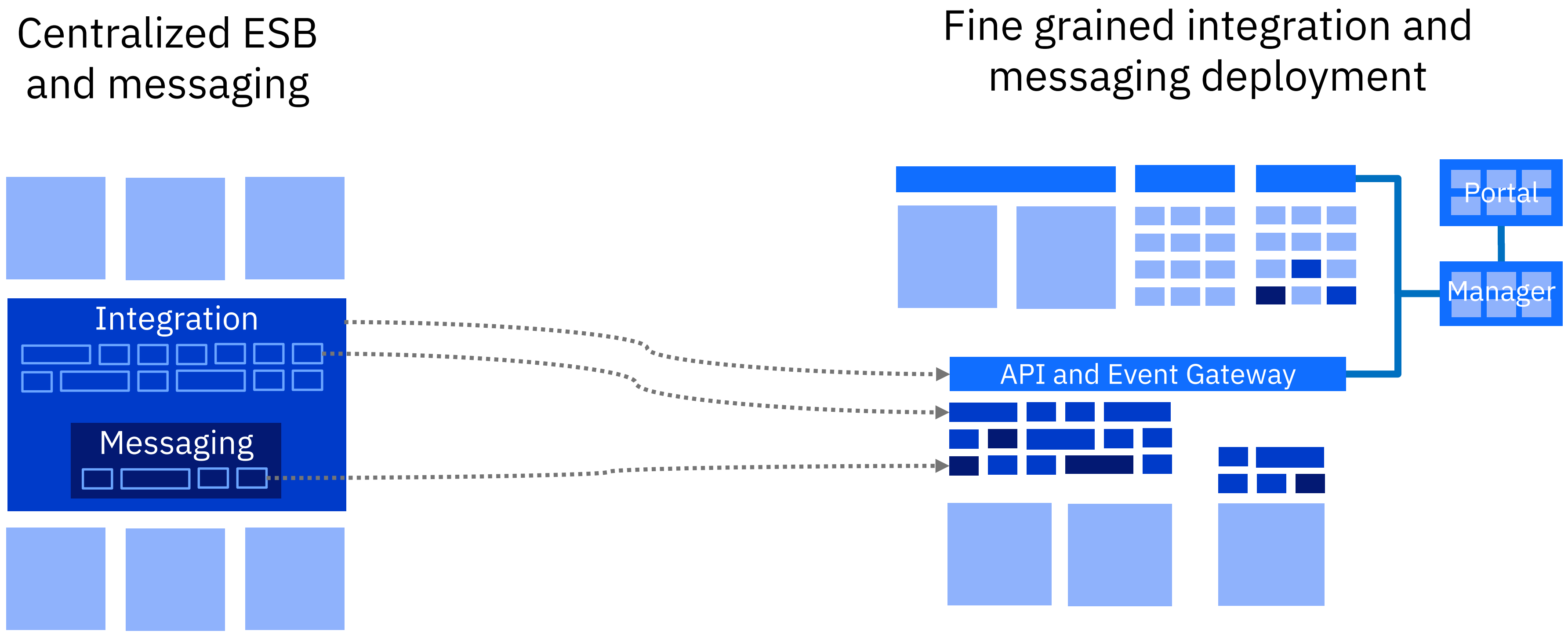
Elastic scalability,
workload orchestration,
cloud infrastructure

Resilience

Minimized dependencies,
discrete failover,
fail fast, start fast

* These are key ***architectural*** aspects of microservices. Clearly a full microservices approach is much broader than this, overlapping heavily with cloud native as a concept

Fine grained deployment, **Appropriate decoupling** and **Minimal state**


Centralized ESB and messaging
Integration
Messaging
Fine grained integration and messaging deployment
Portal
Manager
API and Event Gateway

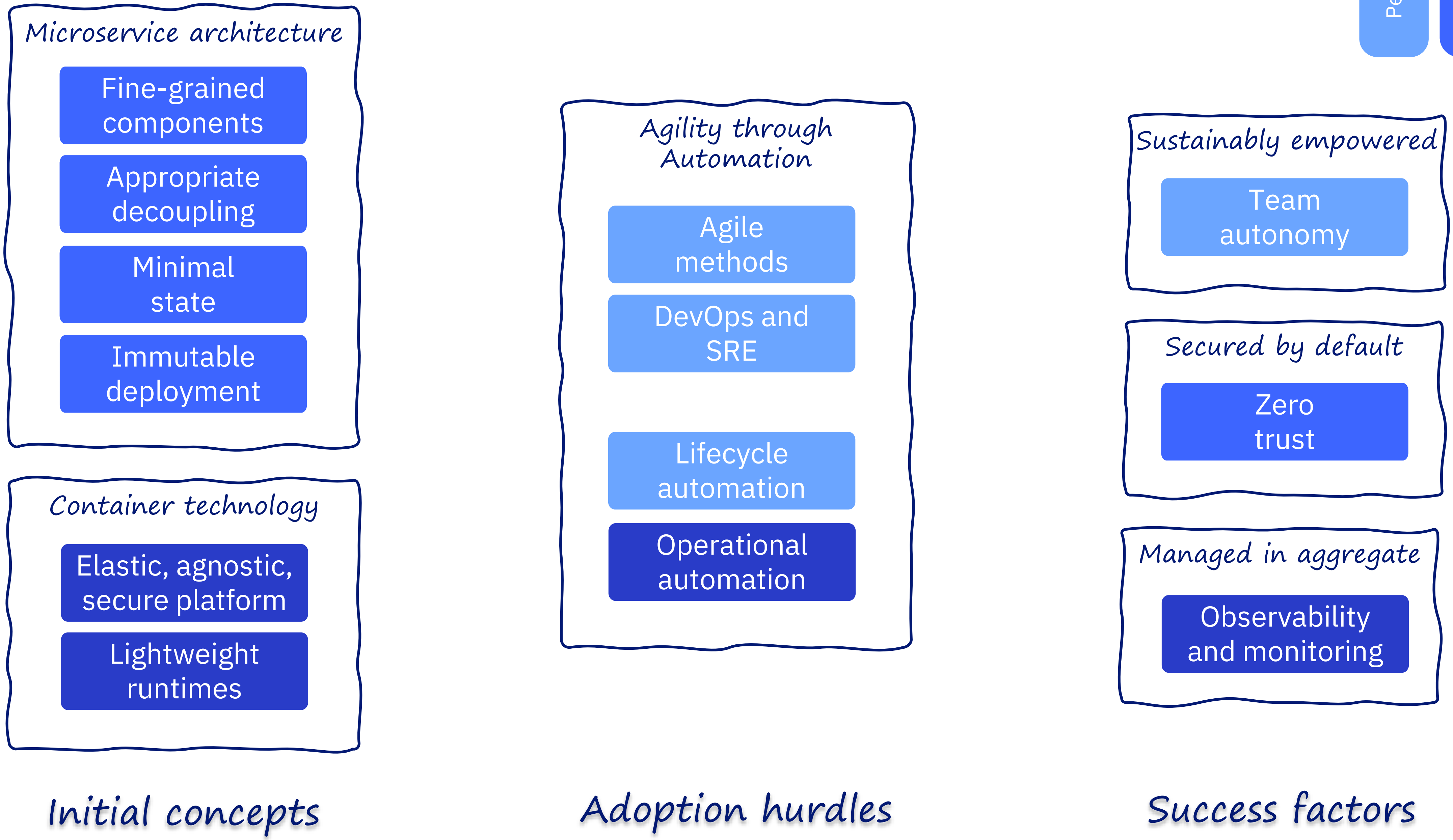
Find grained deployment doesn’t mandate a move to containers, but it will be *easier* in containers

Ingredients of cloud native – an alternative grouping

People

Architecture

Technology



Microservice architecture

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology

- Elastic, agnostic, secure platform
- Lightweight runtimes

Initial concepts

Agility through Automation

- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Adoption hurdles

Sustainably empowered

- Team autonomy

Secured by default

- Zero trust

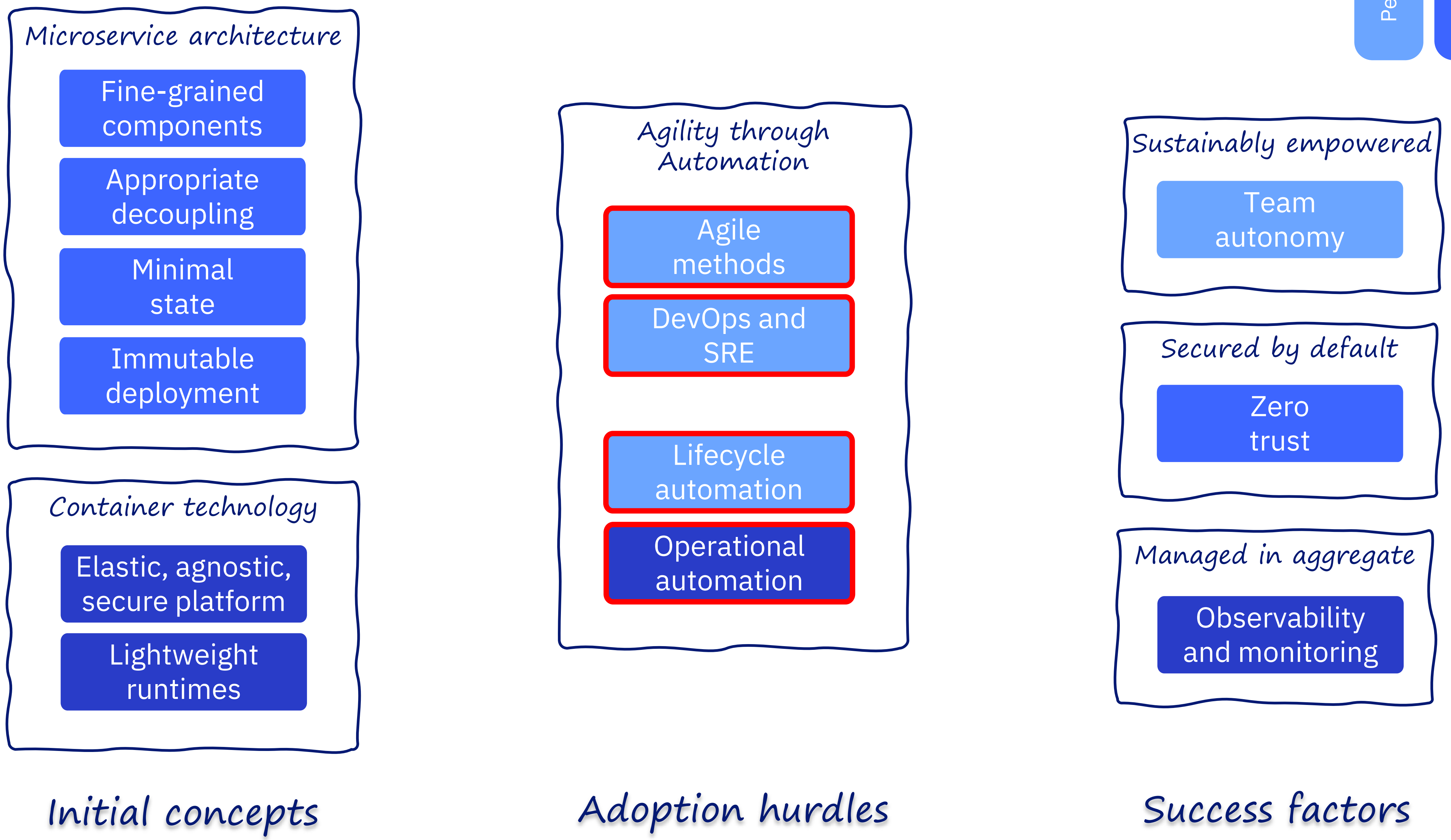
Managed in aggregate

- Observability and monitoring

Success factors

Ingredients of cloud native – an alternative grouping

People | Architecture | Technology



Initial concepts

Microservice architecture
- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology
- Elastic, agnostic, secure platform
- Lightweight runtimes

Adoption hurdles

Agility through Automation
- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Success factors

Sustainably empowered
- Team autonomy

Secured by default
- Zero trust

Managed in aggregate
- Observability and monitoring

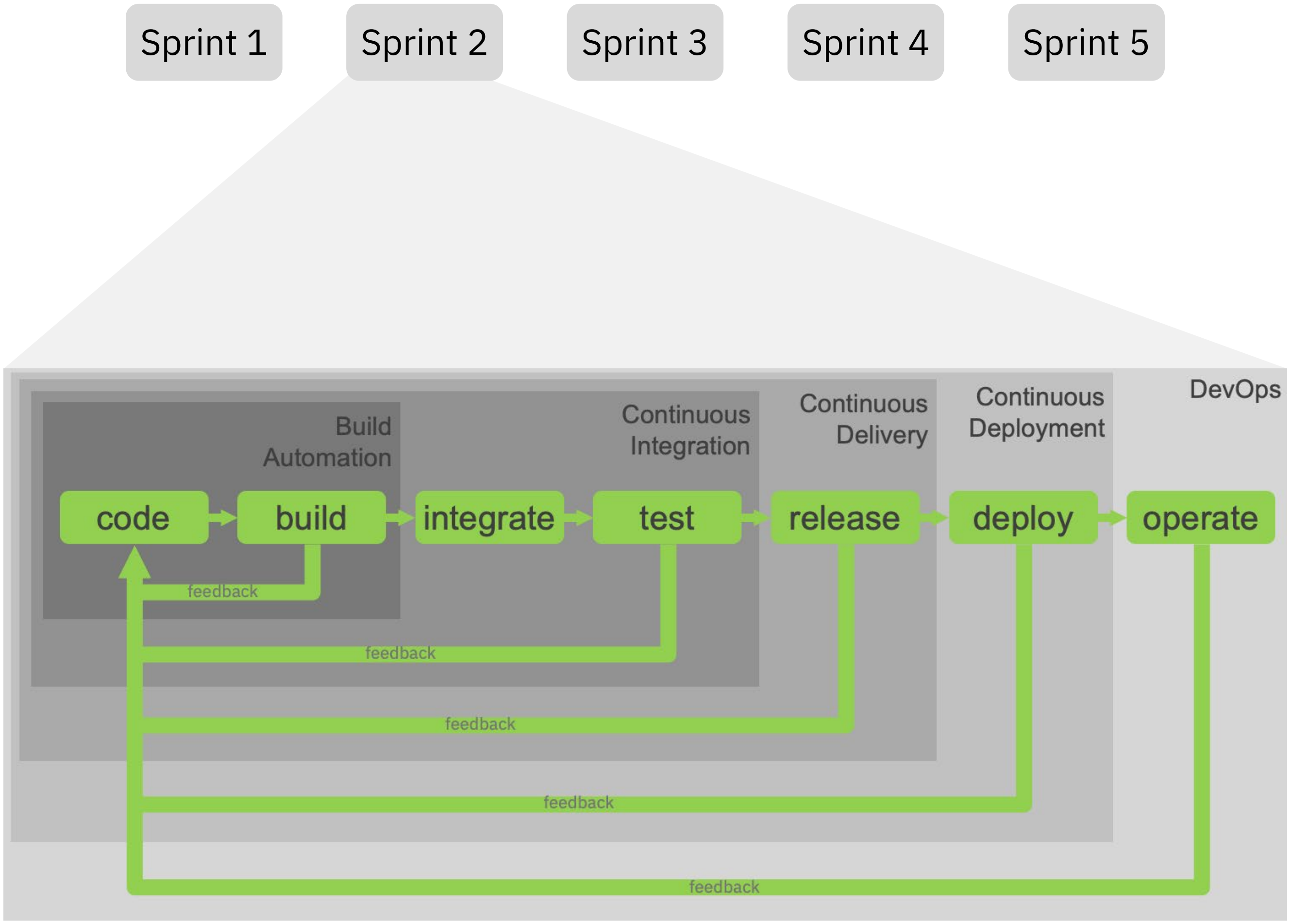
Agility through Automation

Agile methods
- Short, regular iteration cycles.
- Intrinsic business collaboration
- Data driven feedback

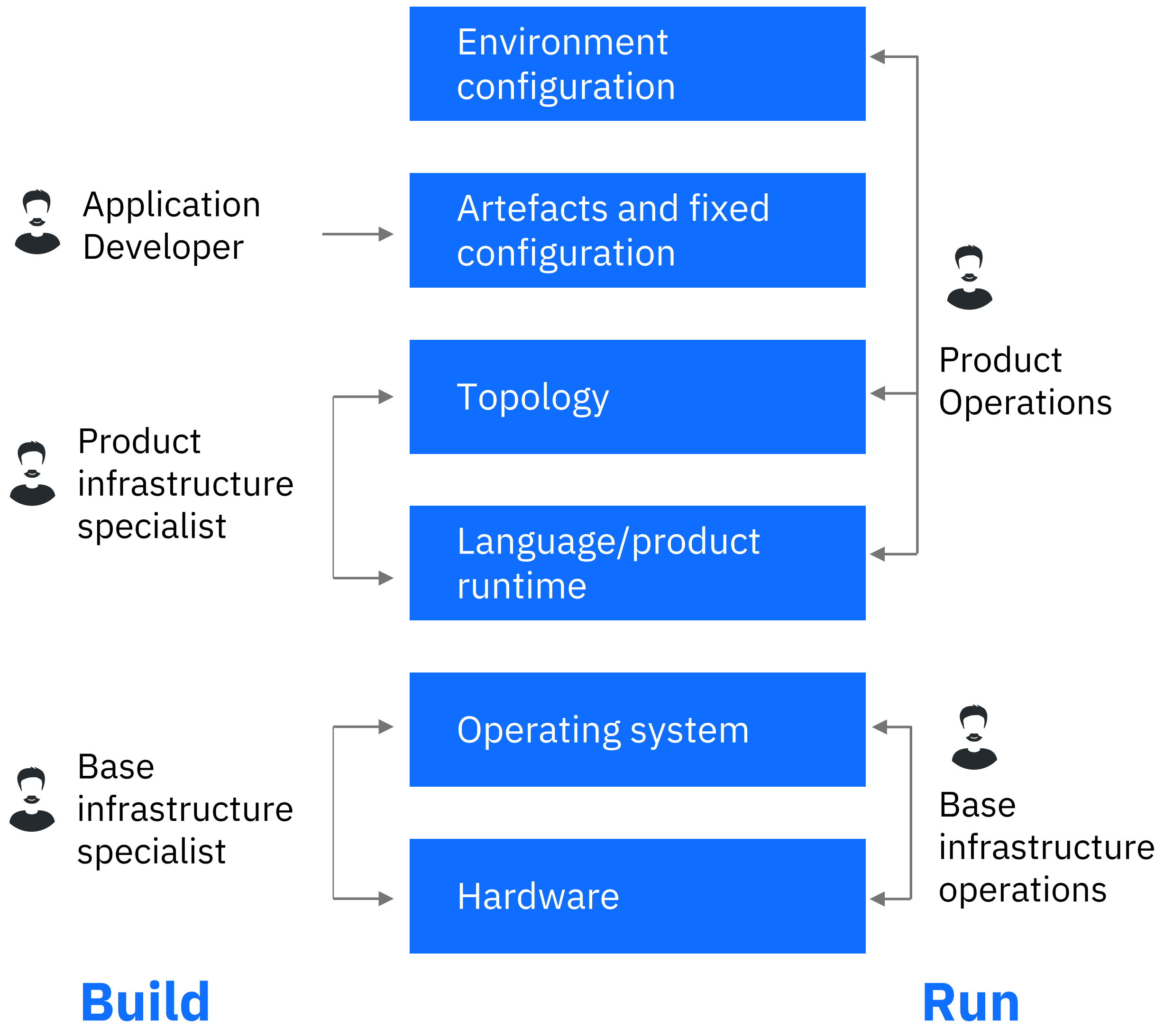
DevOps and site reliability engineering (SRE)
- Collaboration and combination of dev. and ops.
- Shift left for operational concerns
- Rapid operational feedback and resolution

Lifecycle automation
- Continuous Integration – Build/test pipelines
- Continuous Delivery/Deployment – Deploy, verify
- Continuous Adoption – Runtime currency

Operational automation
- Infrastructure as code
- Repository triggered operations (GitOps)
- Site reliability engineering



Traditional



Application Developer

Product infrastructure specialist

Base infrastructure specialist

Environment configuration

Artefacts and fixed configuration

Topology

Language/product runtime

Operating system

Hardware

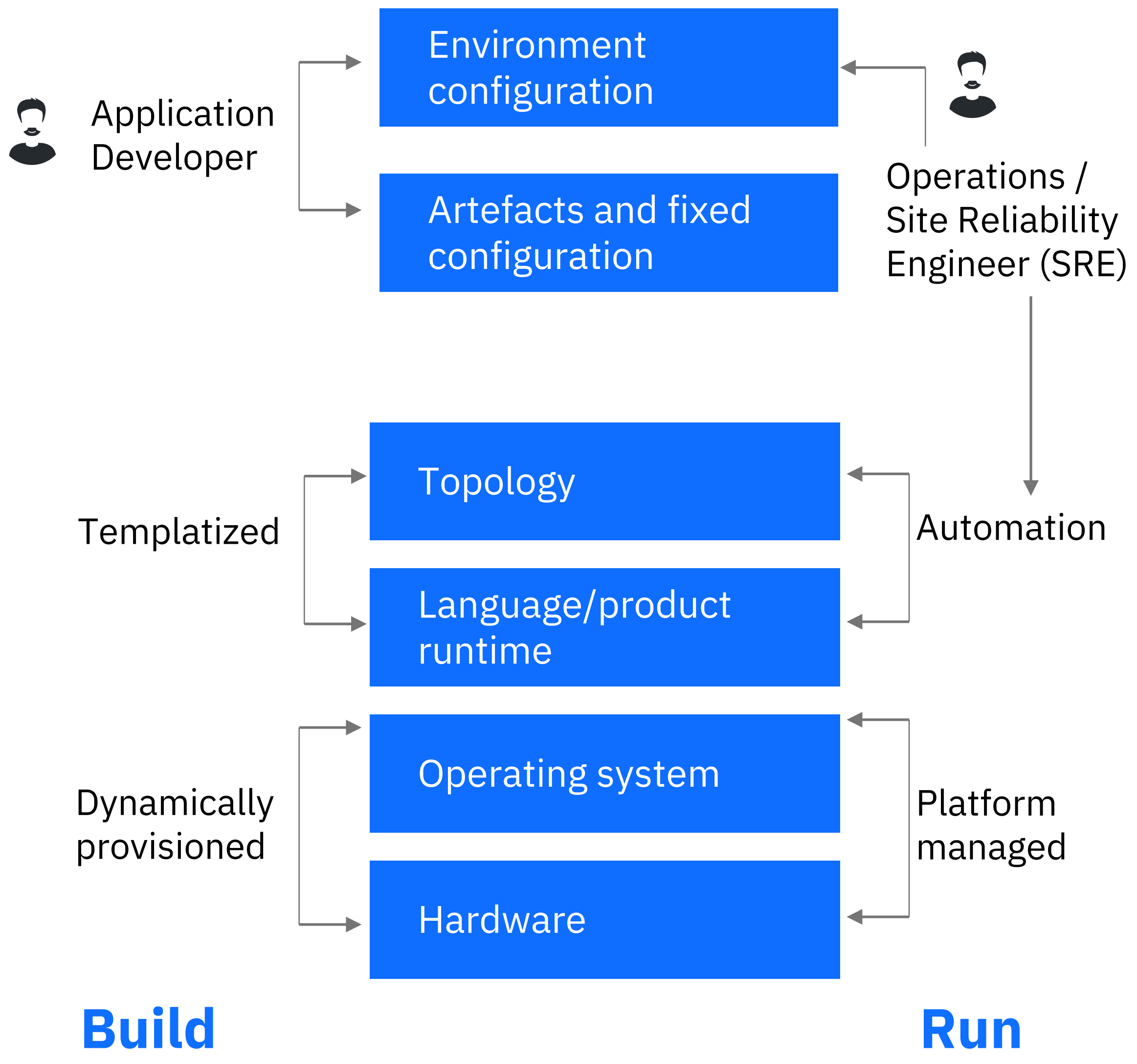
Product Operations

Base infrastructure operations

Build

Run

Cloud native



Application Developer

Templatized

Dynamically provisioned

Environment configuration

Artefacts and fixed configuration

Topology

Language/product runtime

Operating system

Hardware

Operations / Site Reliability Engineer (SRE)

Automation

Platform managed

Build

Run

Agility through Automation

Agile methods

- Short, regular iteration cycles.
- Intrinsic business collaboration
- Data driven feedback

DevOps and site reliability engineering (SRE)

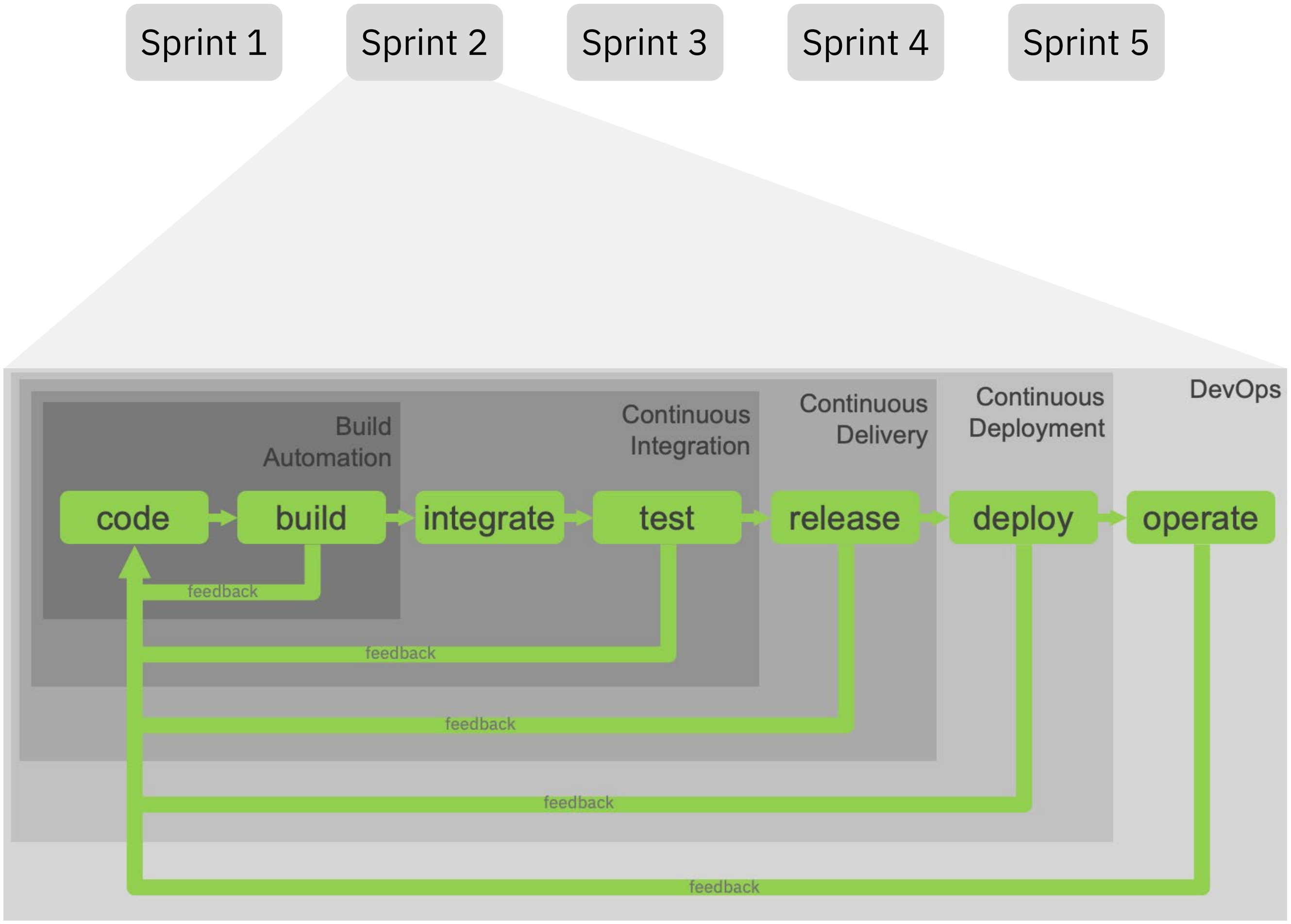
- Collaboration and combination of dev. and ops.
- Shift left for operational concerns
- Rapid operational feedback and resolution

Lifecycle automation

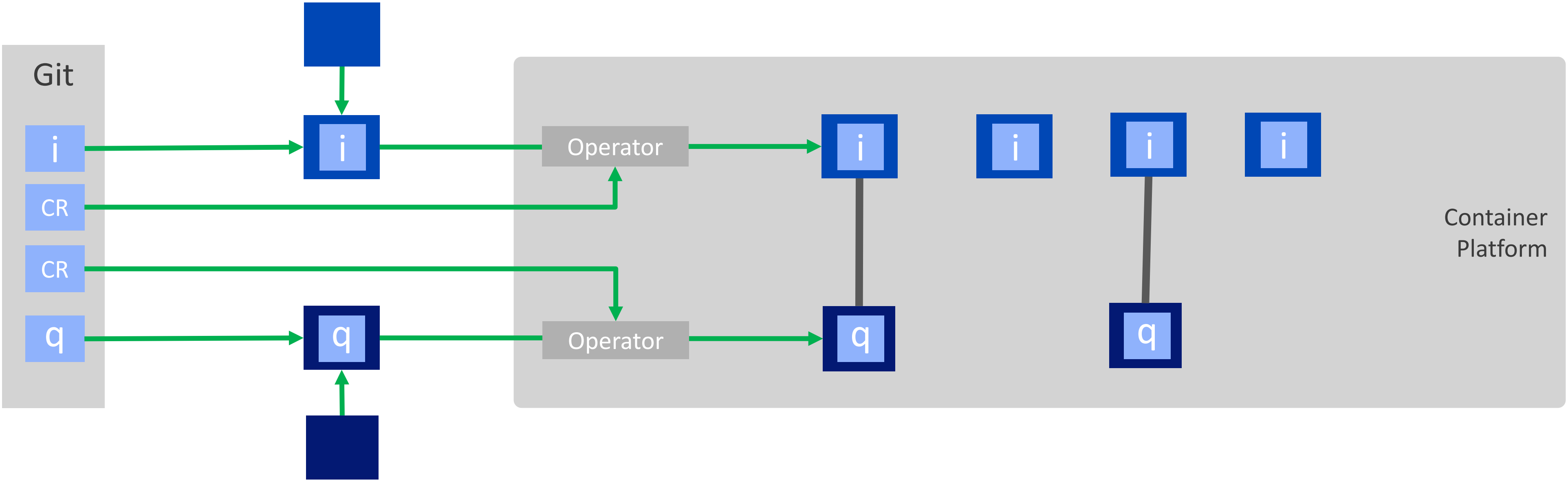
- Continuous Integration – Build/test pipelines
- Continuous Delivery/Deployment – Deploy, verify
- Continuous Adoption – Runtime currency

Operational automation

- Infrastructure as code
- Repository triggered operations (GitOps)
- Site reliability engineering



Lifecycle automation and **Operational automation**



Code automation → **Build automation** → **Deployment automation** → **Operational automation**

Code automation

Code assist

- Flow assembly
- Graphical mapping
- Intelligent connectors
- Pattern templates
- RPA interfaces

Infrastructure as code assist

- Validated form entry
- Guardrails

Build automation

Pipeline(s)

- Git clone
- Dependencies
- Validate
- Package
- Build image
- Test
- Clean up
- Trigger deploy?

Deployment automation

Operator

- Create routes for access
- Provision storage
- Service mesh policy
- Deploys credential
- Wiring to dependencies
- Rollout policy
- Upgrade management
- Multi-part solution deploy?

Operational automation

Operator

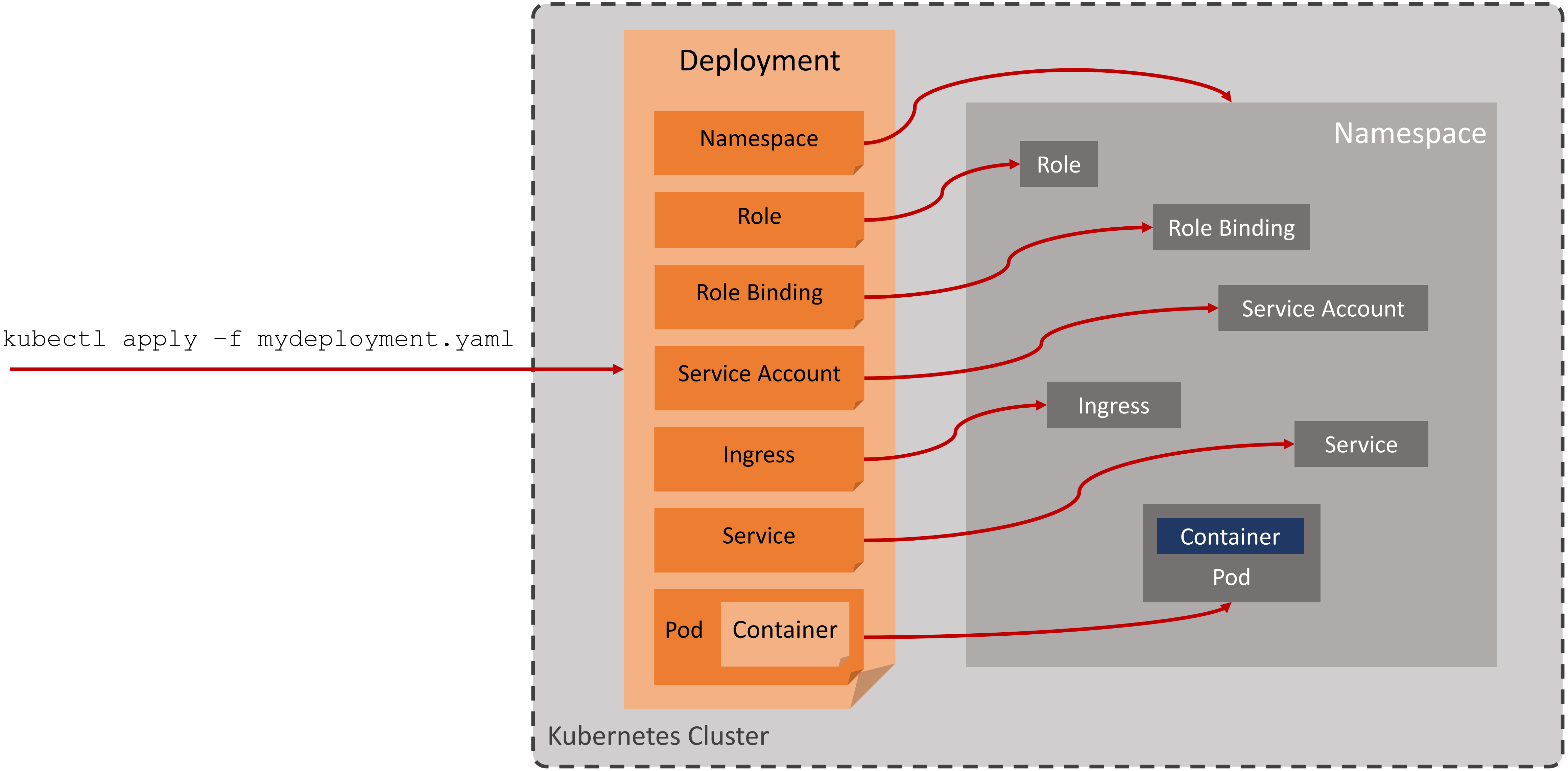
- HA/auto recovery
- Auto scaling
- Log collation and interpretation
- Alerts

🟦	Integration runtime image	■	Messaging runtime image
i	Integration code	q	Queue manager definition
OAS	Open API Specification	CRD	Custom resource

Some of the Kubernetes objects involved in a deployment

To deploy a container into Kubernetes, you have to define these!

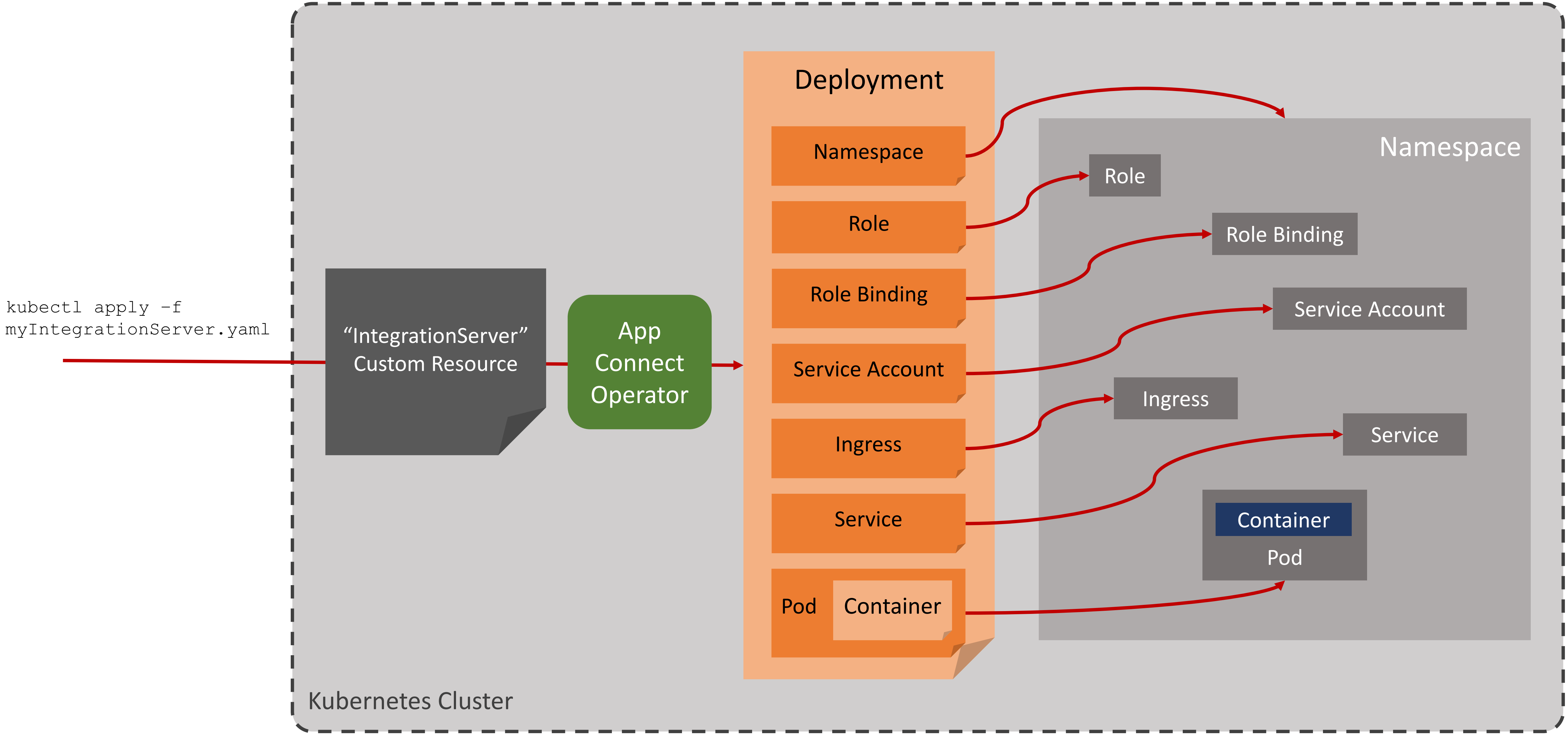
```
kubectl apply -f mydeployment.yaml
```



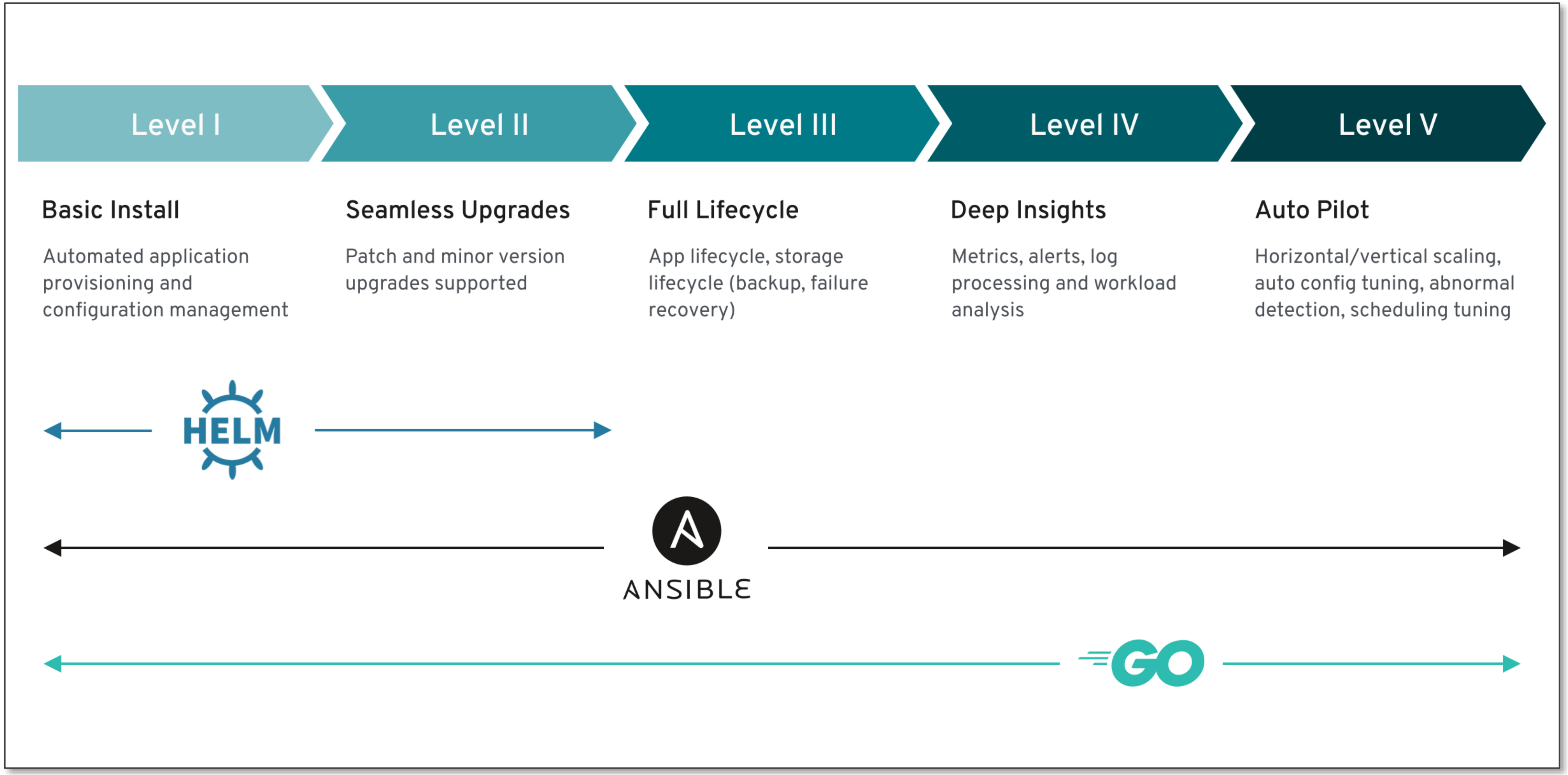
The role of a Kubernetes “Operator”

Translate your requirements (custom resource) into Kubernetes objects, instantiate them, and look after them

```
kubectl apply -f
myIntegrationServer.yaml
```



Operator maturity model



Level I

Basic Install

Automated application provisioning and configuration management

Level II

Seamless Upgrades

Patch and minor version upgrades supported

Level III

Full Lifecycle

App lifecycle, storage lifecycle (backup, failure recovery)

Level IV

Deep Insights

Metrics, alerts, log processing and workload analysis

Level V

Auto Pilot

Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

Source: https://docs.openshift.com/container-platform/4.9/operators/understanding/olm-what-operators-are.html#olm-maturity-model_olm-what-operators-are

Example definition of an **`IntegrationServer`** custom resource object

This yaml file instructs the App Connect Operator to

- **Deploy a single replica of the IBM App Connect Certified Container, allocating it 1/3 of a CPU, and making the container available via HTTP**
- Pull down a bar file from a remote location, and load it on start up

The Operator will translate those requirements into all the necessary Kubernetes objects

```
apiVersion: appconnect.ibm.com/v1beta1
kind: IntegrationServer
metadata:
  name: http-echo-service
  namespace: ace-demo
  labels: {}
spec:
  adminServerSecure: false
  barURL: >-
    https://github.com/amarIBM/hello-world/raw/master/HttpEchoApp.bar
  configurations:
    - github-barauth
  createDashboardUsers: true
  designerFlowsOperationMode: disabled
  enableMetrics: true
  license:
    accept: true
    license: L-KSBM-C37J2R
    use: AppConnectEnterpriseProduction
  pod:
    containers:
      runtime:
        resources:
          limits:
            cpu: 300m
            memory: 350Mi
          requests:
            cpu: 300m
            memory: 300Mi
  replicas: 1
  router:
    timeout: 120s
  service:
    endpointType: http
  version: '12.0'
```

https://community.ibm.com/community/user/integration/blogs/amar-shah1/2021/11/21/deploy-a-simple-flow-onto-red-hat-openshift-using

Example definition of an **IntegrationServer** custom resource object

This yaml file instructs the App Connect Operator to

- Deploy a single replica of the IBM App Connect Certified Container, allocating it 1/3 of a CPU, and making the container available via HTTP

- **Pull down a bar file from a remote location, and load it on start up**

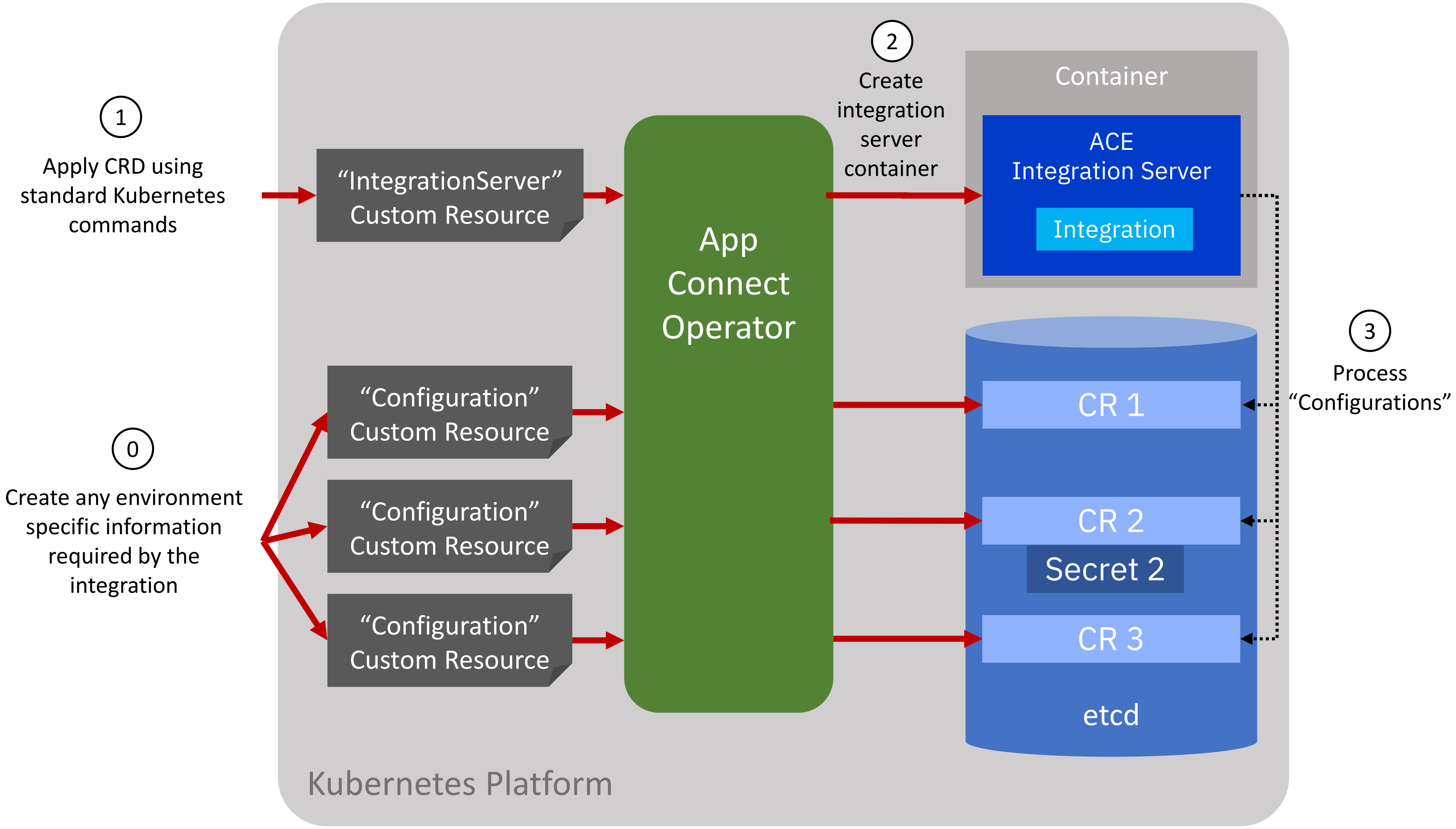
The Operator will translate those requirements into all the necessary Kubernetes objects

```yaml
apiVersion: appconnect.ibm.com/v1beta1
kind: IntegrationServer
metadata:
  name: http-echo-service
  namespace: ace-demo
  labels: {}
spec:
  adminServerSecure: false
  barURL: >-
    https://github.com/amarIBM/hello-world/raw/master/HttpEchoApp.bar
  configurations:
    - github-barauth
  createDashboardUsers: true
  designerFlowsOperationMode: disabled
  enableMetrics: true
  license:
    accept: true
    license: L-KSBM-C37J2R
    use: AppConnectEnterpriseProduction
  pod:
    containers:
      runtime:
        resources:
          limits:
            cpu: 300m
            memory: 350Mi
          requests:
            cpu: 300m
            memory: 300Mi
  replicas: 1
  router:
    timeout: 120s
  service:
    endpointType: http
  version: '12.0'
```

https://community.ibm.com/community/user/integration/blogs/amar-shah1/2021/11/21/deploy-a-simple-flow-onto-red-hat-openshift-using

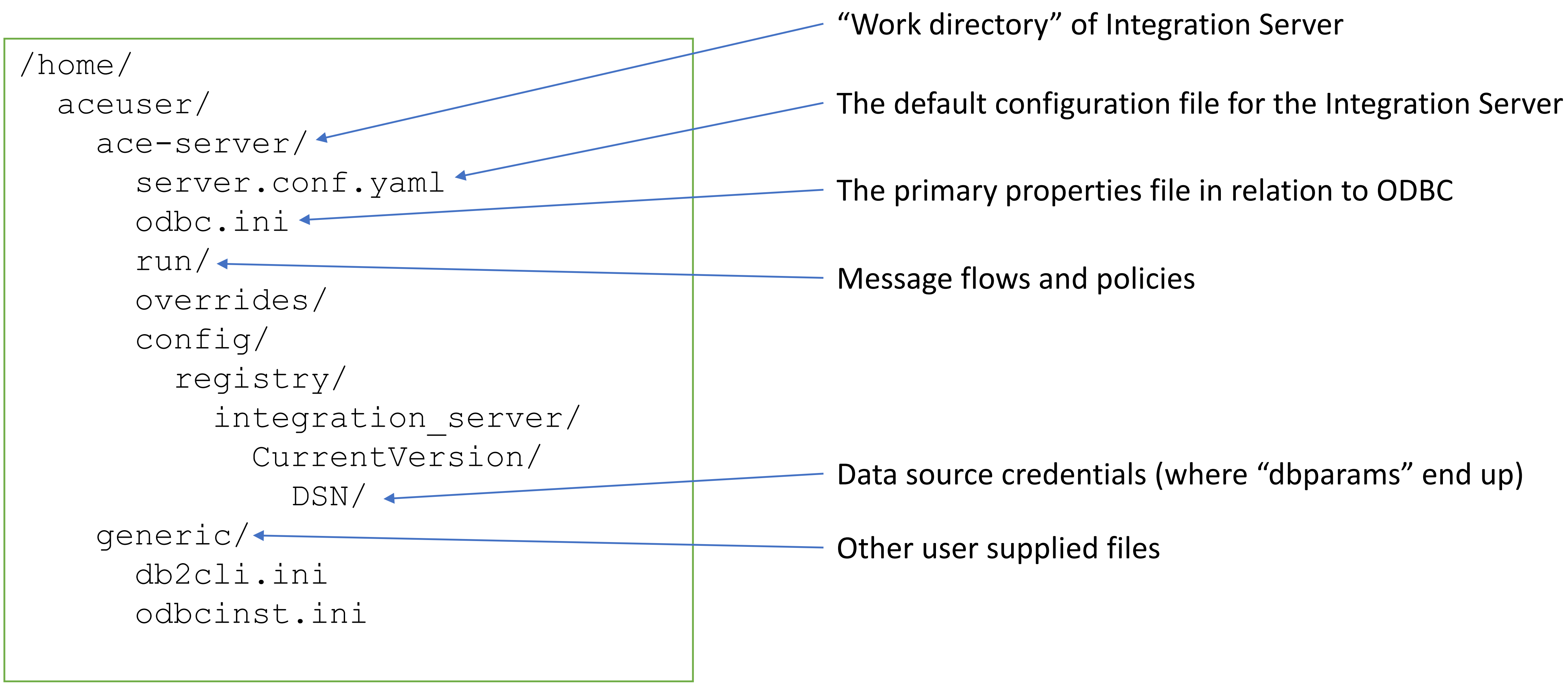
ACE “Configurations”

The ACE Operator provides an abstraction from how configuration is stored and processed



1
Apply CRD using standard Kubernetes commands
0
Create any environment specific information required by the integration
“IntegrationServer” Custom Resource
“Configuration” Custom Resource
“Configuration” Custom Resource
“Configuration” Custom Resource
App Connect Operator
2
Create integration server container
Container
ACE Integration Server
Integration
3
Process “Configurations”
CR 1
CR 2
Secret 2
CR 3
etcd
Kubernetes Platform

What does the certified container actually do with the “Configurations”?
(using a connection to an ODBC database as an example)



```
/home/
  aceuser/
    ace-server/
      server.conf.yaml
      odbc.ini
      run/
      overrides/
      config/
        registry/
          integration_server/
            CurrentVersion/
              DSN/
    generic/
      db2cli.ini
      odbcinst.ini
```

“Work directory” of Integration Server

The default configuration file for the Integration Server

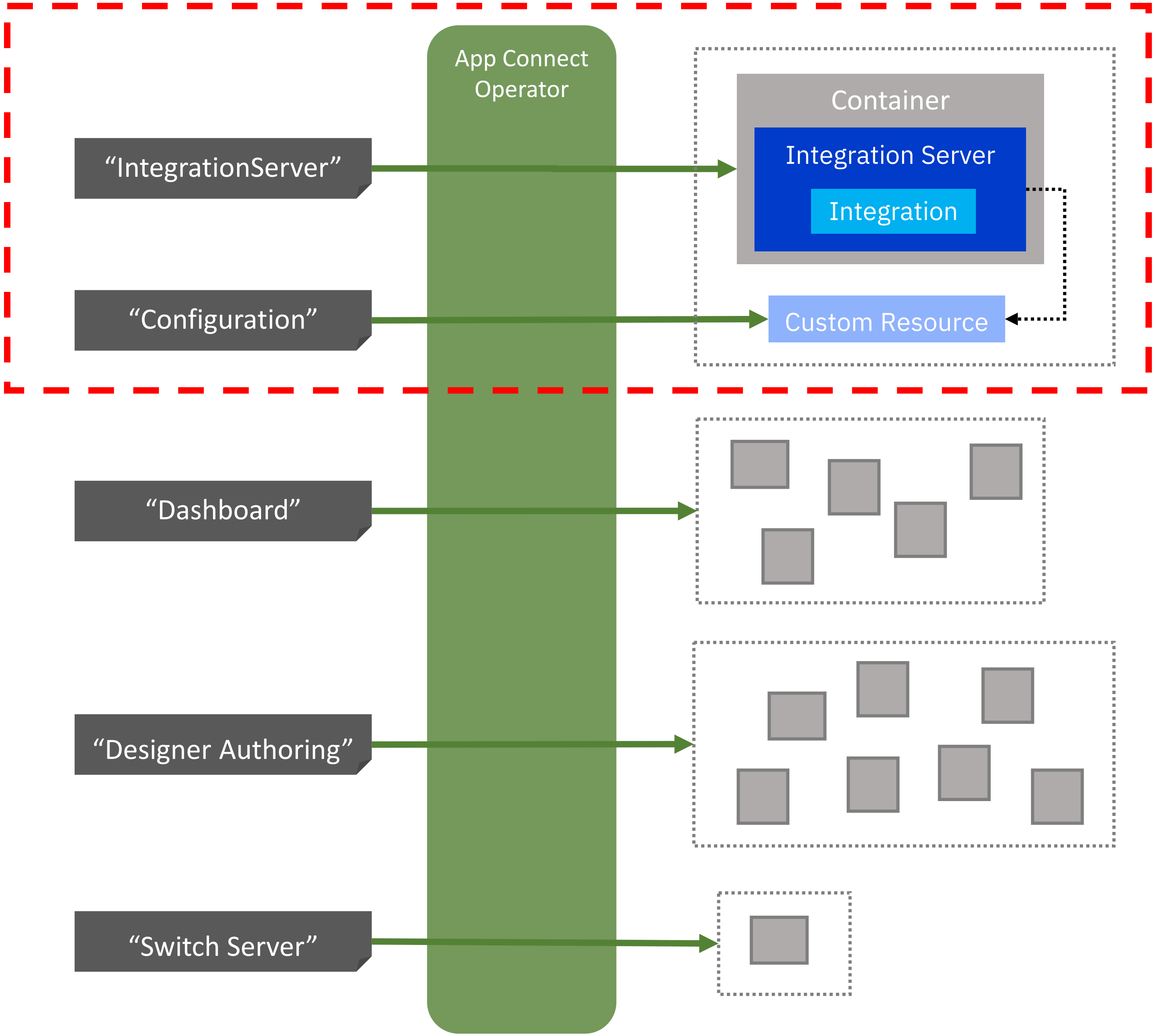
The primary properties file in relation to ODBC

Message flows and policies

Data source credentials (where “dbparams” end up)

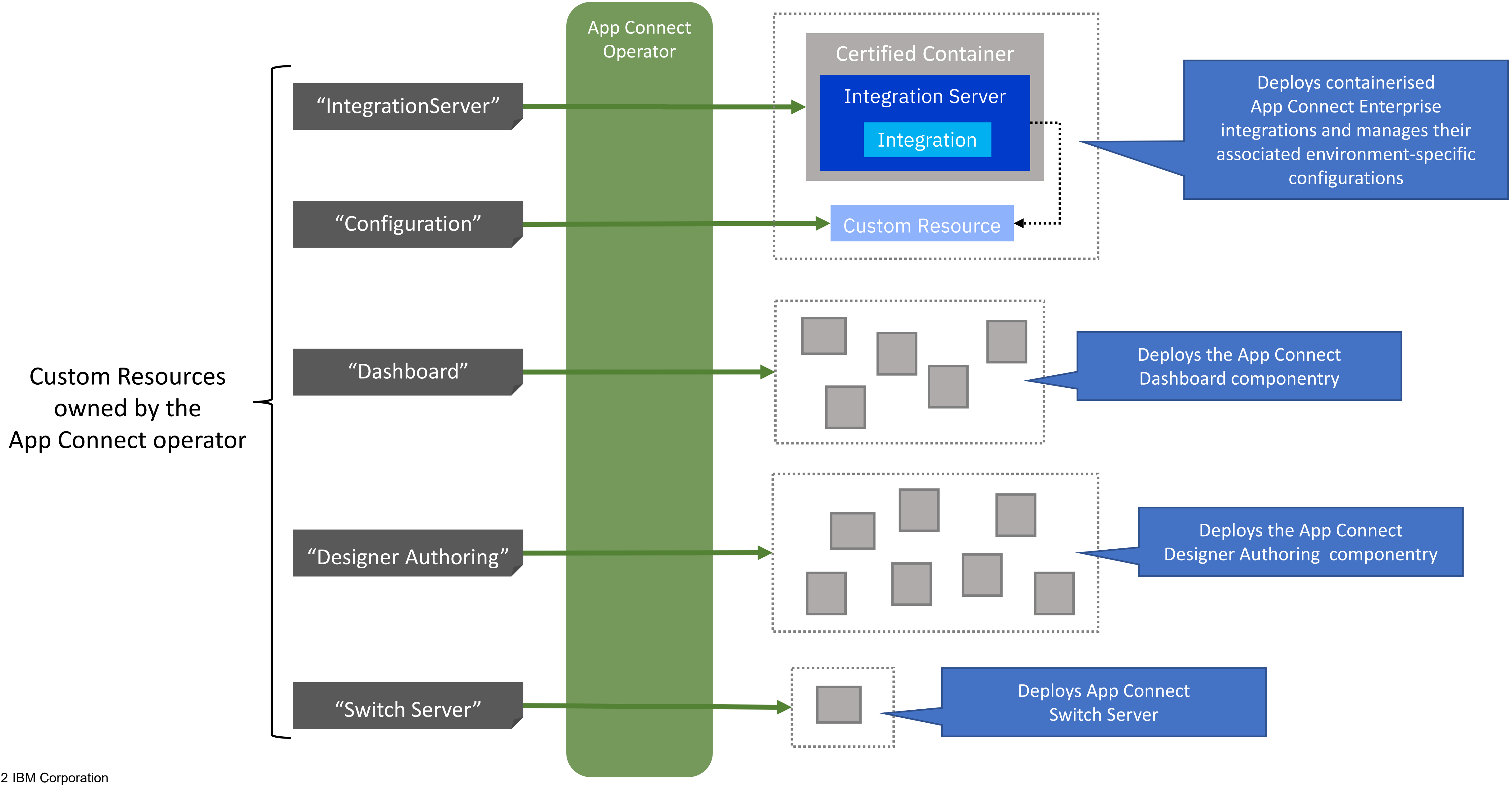
Other user supplied files

This presentation will focus on one function of the Operator for IBM App Connect for illustration purposes



© 2022 IBM Corporation

What does the App Connect Operator do?

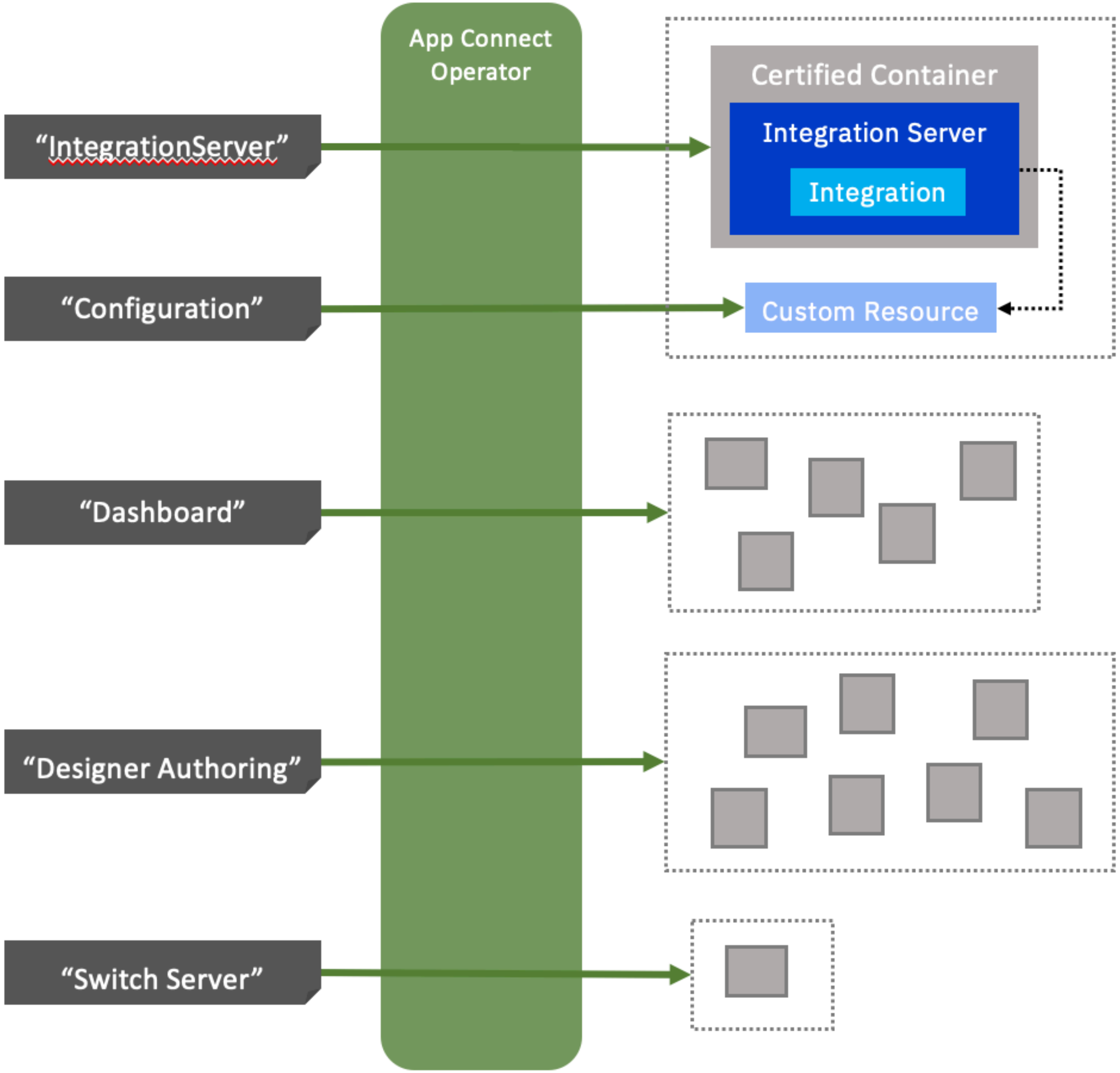


Deploys containerised App Connect Enterprise integrations and manages their associated environment-specific configurations

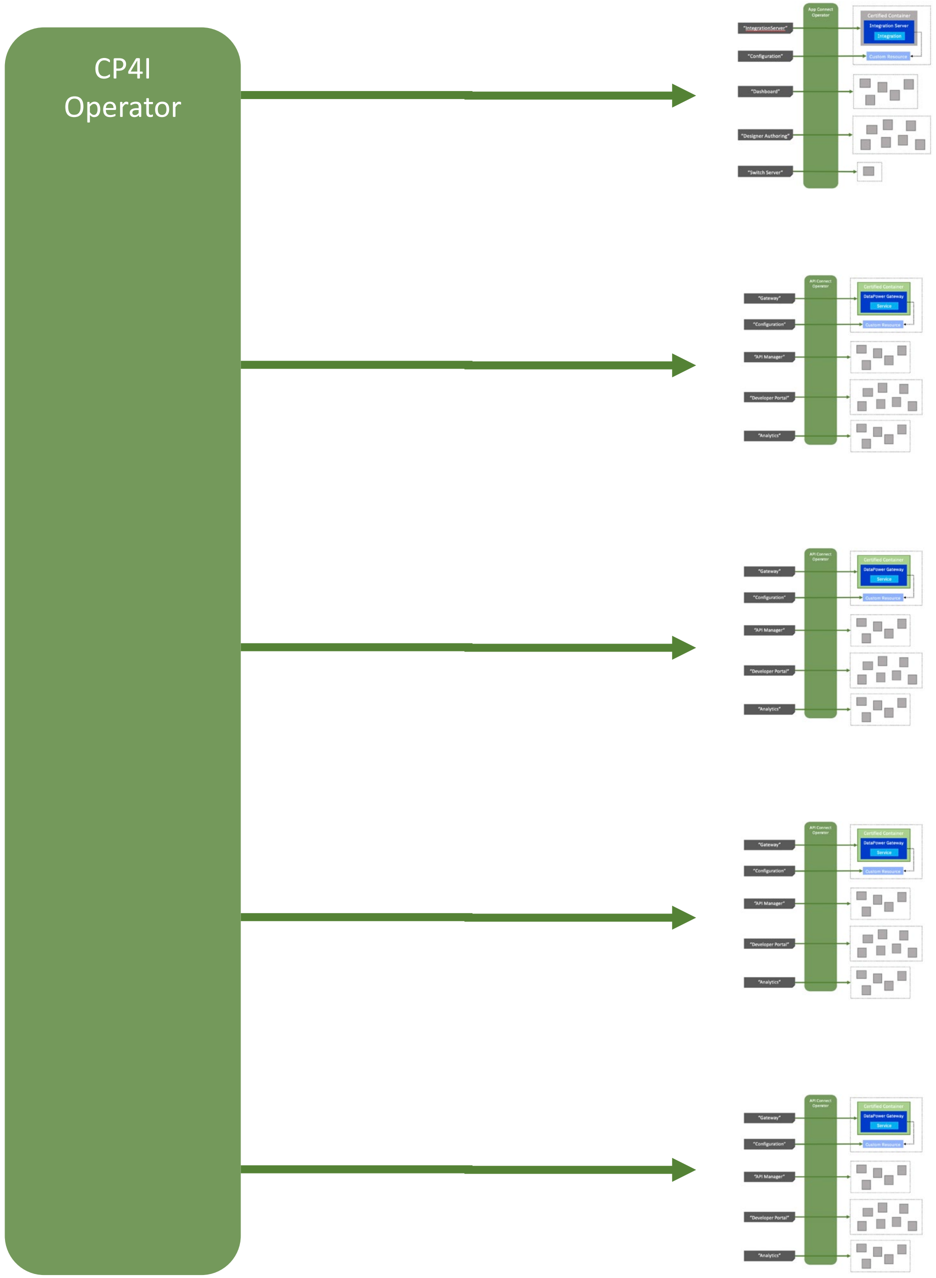
Deploys the App Connect Dashboard componentry

Deploys the App Connect Designer Authoring componentry

Deploys App Connect Switch Server


App Connect Operator
"IntegrationServer"
"Configuration"
"Dashboard"
"Designer Authoring"
"Switch Server"
Certified Container
Integration Server
Integration
Custom Resource

Cloud Pak for Integration Operator



CP4I Operator

App Connect Operator

API Connect Operator

MQ Operator

Event Streams Operator

Aspera Operator

Blog series: “From IBM Integration Bus to App Connect Enterprise in containers”

Scenario 1: Deploying a simple flow on Docker
Introduces the App Connect certified container

Scenario 2: Deploying a simple flow on Red Hat OpenShift
Introduces Operators, and Configuration objects, and App Connect Dashboard

Scenario 3: Load balancing and autoscaling a simple App Connect flow
Discusses Kubernetes replication. Introduces "pods".

Scenario 4: Deploying an IBM MQ queue manager in a container
Introduces ConfigMaps and Secrets

Scenario 5: Moving an App Connect flow using MQ onto containers
Explores separation of MQ from ACE, and how to perform policy overrides

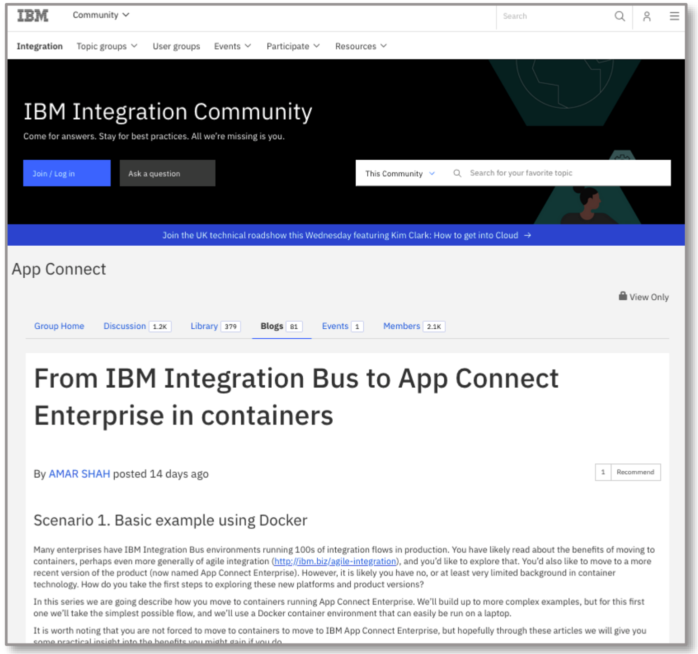
Scenario 6: Moving an App Connect flow that connects to a database onto containers
Shows the action of the Operator with multiple different Configuration types

Scenario 7: Deploying an App Connect integration on Amazon EKS
Defines the additional steps necessary to use Operators on non-OpenShift environments

The IBM App Connect Operator
Part 1 - What is an Operator and why did we create one for IBM App Connect?
Part 2 - Exploring the IntegrationServer resource of the IBM App Connect Operator
Getting Practical with Operators in IBM App Connect (webinar from TechCon 2022)

Container deployment
Comparing styles of container deployment for IBM App Connect (a.k.a baked vs fried!)

…many more coming…



IBM Integration Community

Come for answers. Stay for best practices. All we’re missing is you.

Join the UK technical roadshow this Wednesday featuring Kim Clark: How to get into Cloud →

App Connect

From IBM Integration Bus to App Connect Enterprise in containers

By AMAR SHAH posted 14 days ago

Scenario 1. Basic example using Docker

Many enterprises have IBM Integration Bus environments running 100s of integration flows in production. You have likely read about the benefits of moving to containers, perhaps even more generally of agile integration (http://ibm.biz/agile-integration), and you’d like to explore that. You’d also like to move to a more recent version of the product (now named App Connect Enterprise). However, it is likely you have no, or at least very limited background in container technology. How do you take the first steps to exploring these new platforms and product versions?

In this series we are going describe how you move to containers running App Connect Enterprise. We’ll build up to more complex examples, but for this first one we’ll take the simplest possible flow, and we’ll use a Docker container environment that can easily be run on a laptop.

It is worth noting that you are not forced to move to containers to move to IBM App Connect Enterprise, but hopefully through these articles we will give you some practical insight into the benefits you might gain if you do

Many more coming, watch this space!
http://ibm.biz/iib-ace - **please do make suggestions on further topics in the comments.**

Articles in progress:

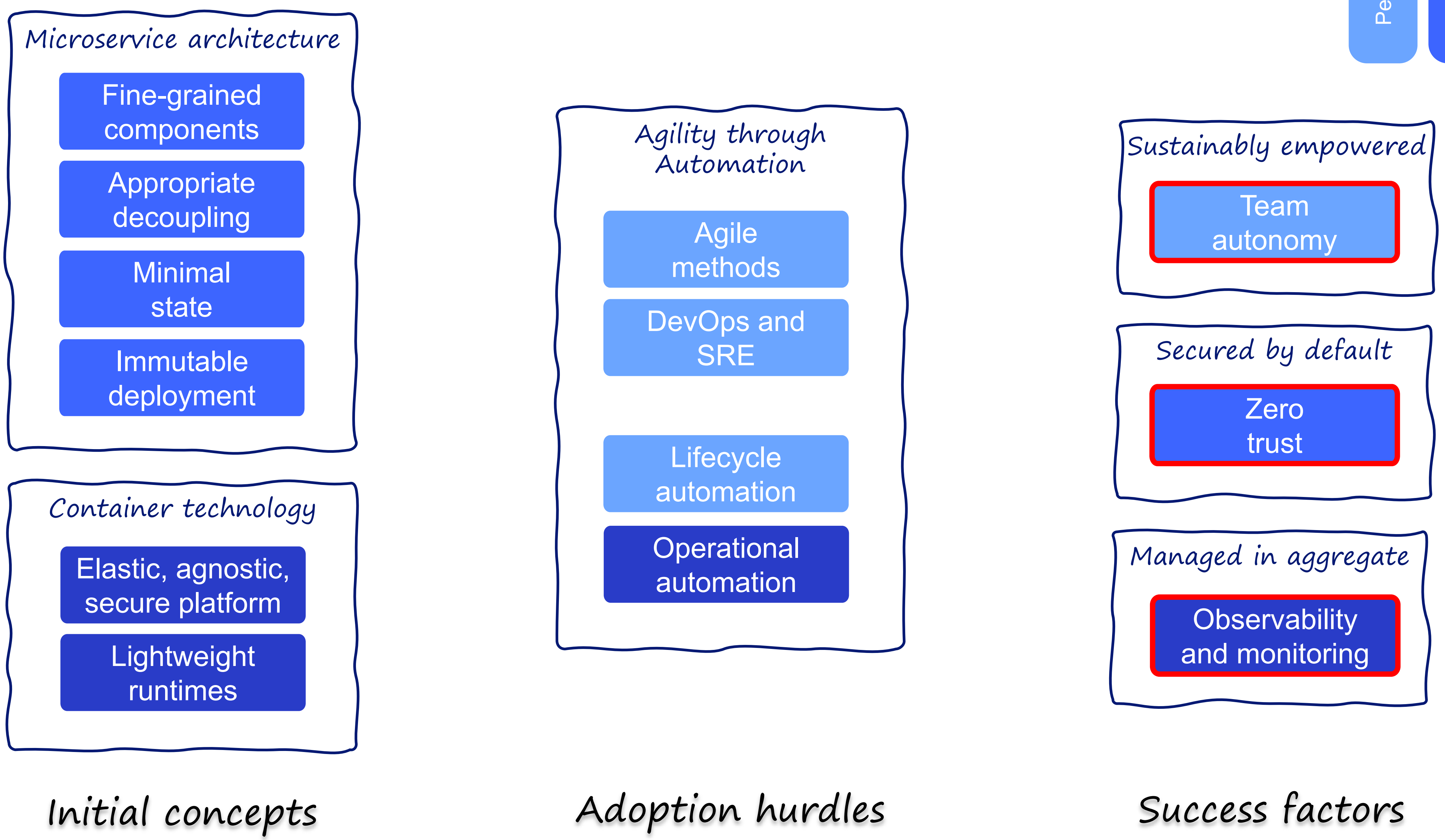
- Fried vs baked deployment
- ACE CICD pipelines

Ingredients of cloud native – an alternative grouping

People

Architecture

Technology



Microservice architecture

- Fine-grained components
- Appropriate decoupling
- Minimal state
- Immutable deployment

Container technology

- Elastic, agnostic, secure platform
- Lightweight runtimes

Initial concepts

Agility through Automation

- Agile methods
- DevOps and SRE
- Lifecycle automation
- Operational automation

Adoption hurdles

Sustainably empowered

- Team autonomy

Secured by default

- Zero trust

Managed in aggregate

- Observability and monitoring

Success factors

Sustainably empowered

Team autonomy

- Decentralized ownership
- Technological freedom
- Self-provisioning

Success factors

Secured by default

Zero trust

- Minimized privileges
- Implicit data security
- Shift Left for security (DevSecOps)

Managed in aggregate

Observability and monitoring

- Easily accessible status
- Platform neutral logging and tracing
- Cross component correlation

What do we mean by Zero Trust* in the context of **this** presentation?

"Zero trust (ZT) is the term for an evolving set of cybersecurity paradigms that move defenses from static, network-based perimeters to focus on users, assets, and resources…."

NIST – Zero Trust Architecture (2020)

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

Approaches/strategies

Threat modelling

Think like a hacker

Defense in depth

Buzz phrases

- Identity as a perimeter
- Micro segmentation
- Adaptive security
- …

Themes

- Assume any vulnerability will be exploited
- Don't trust anyone or anything
- Assume attackers are on the inside already

* The term "zero trust" in computing has actually been around since at least 1994, but the concept and details have evolved significantly over time.

Zero trust

Minimized privileges

- Components and people should have no privileges by default
- All privileges are explicitly bestowed based on identity

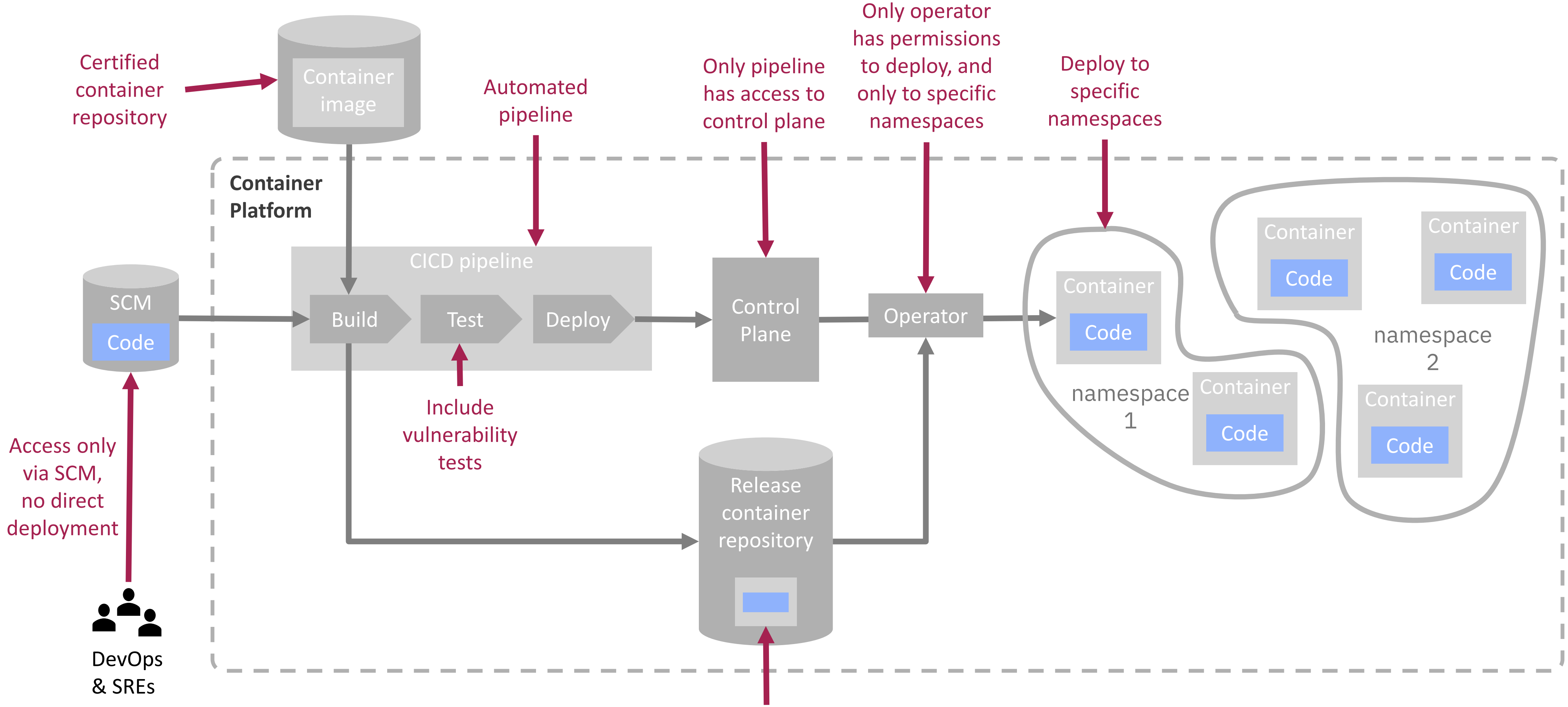
Implicit data security

- Data should always be safe, whether at rest or in transit
- Data access control should be identity based

Shift Left for security (DevSecOps)

- Security should be included at the earliest point in the lifecycle
- All environments are vulnerable, not just production

How do you avoid bad code entering the system in the first place?



Certified container repository

Automated pipeline

Only pipeline has access to control plane

Only operator has permissions to deploy, and only to specific namespaces

Deploy to specific namespaces

Include vulnerability tests

Access only via SCM, no direct deployment

Immutable deployment image

But zero trust principles mean you can't assume that even this is sufficient!

*Some** perspectives on Zero Trust

*(*this is far from an exhaustive list)*

1. **Identity** *as a perimeter*
2. **Privileges** should be *minimized*
3. **Data** must *always* be safe
4. **Secrets**...are *secret*

Cloud Native

http://ibm.biz/cloudnativedefined
https://ibm.biz/agile-integration-cloud-native

Agile Integration

http://ibm.biz/agile-integration
http://ibm.biz/agile-integration-webinar
http://ibm.biz/cp4i-security-webinar
http://ibm.biz/agile-integration-webcasts

Specific topic webinars from TechCon

- Operators
- Pipelines
- Zero Trust

Blog series: Moving to App Connect Enterprise in containers
http://ibm.biz/iib-ace

Other key links on agile integration

http://ibm.biz/agile-integration-links

Staying up to date:

https://community.ibm.com/community/user/integration

IBM Integration

https://developer.ibm.com/integration

Cloud Pak for Integration

https://www.ibm.com/cloud/cloud-pak-for-integration

Thank you.

IBM