Welkom

WAZUG.NL

AZURE USER GROUP NL

Enterprise Portfolio Modernization at Scale

Modernize your business to get ahead and stay ahead

Wazug
Clemens Reijnen

Cloud Benefits

Agility

Experiment

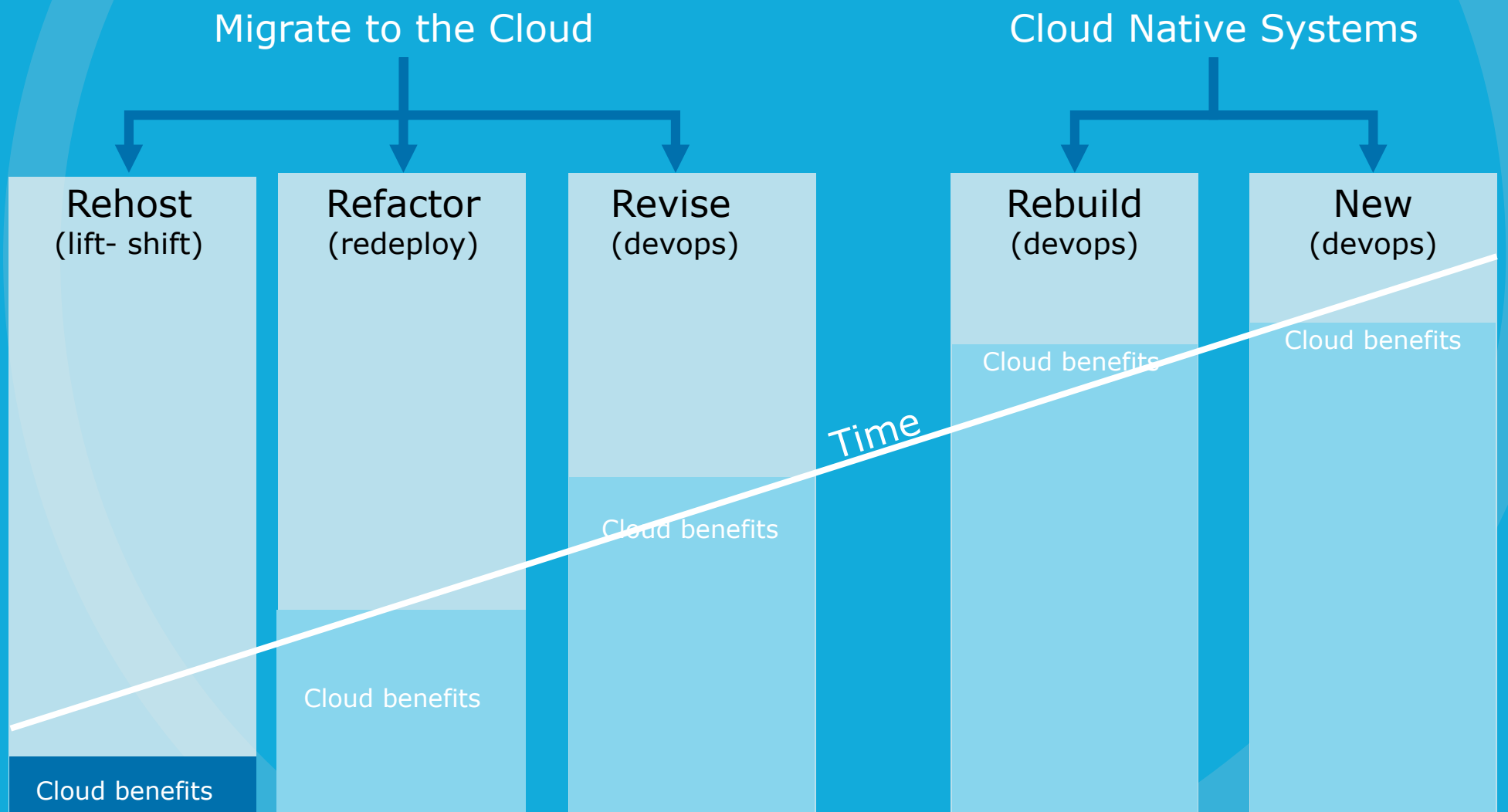
Cost

Elasticity

Recovery

Security

Cloud adoption



Enterprises must think on…

Self organizing teams

Target Operating Model

Regulations

Cloud benefits

Maintainability

Consistency at scale

Skills

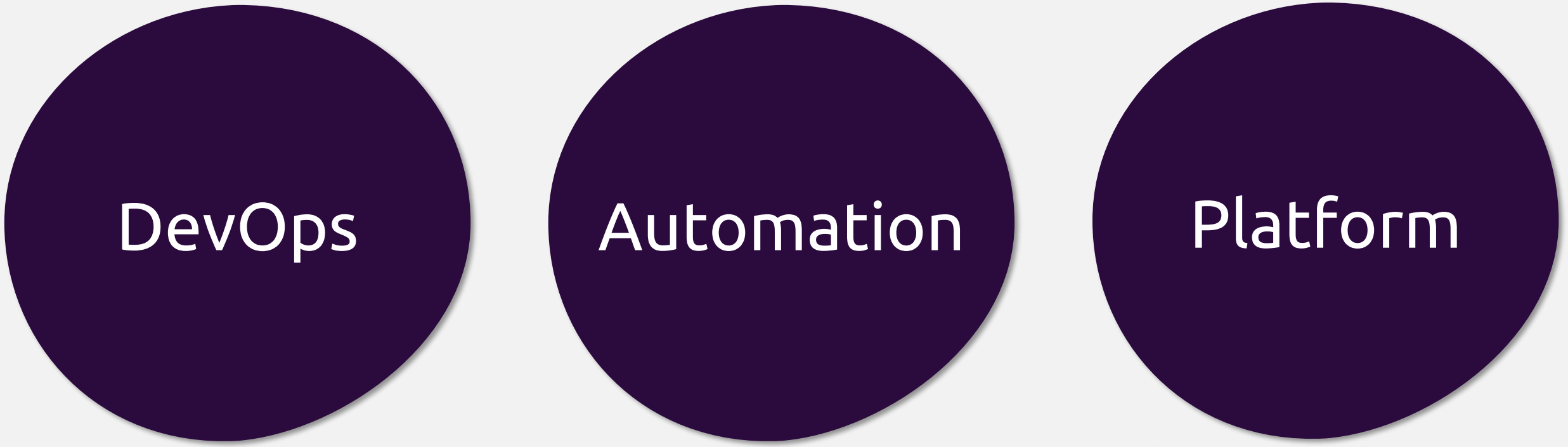
Governance

Application portfolio

Compliancy

Security

Focus



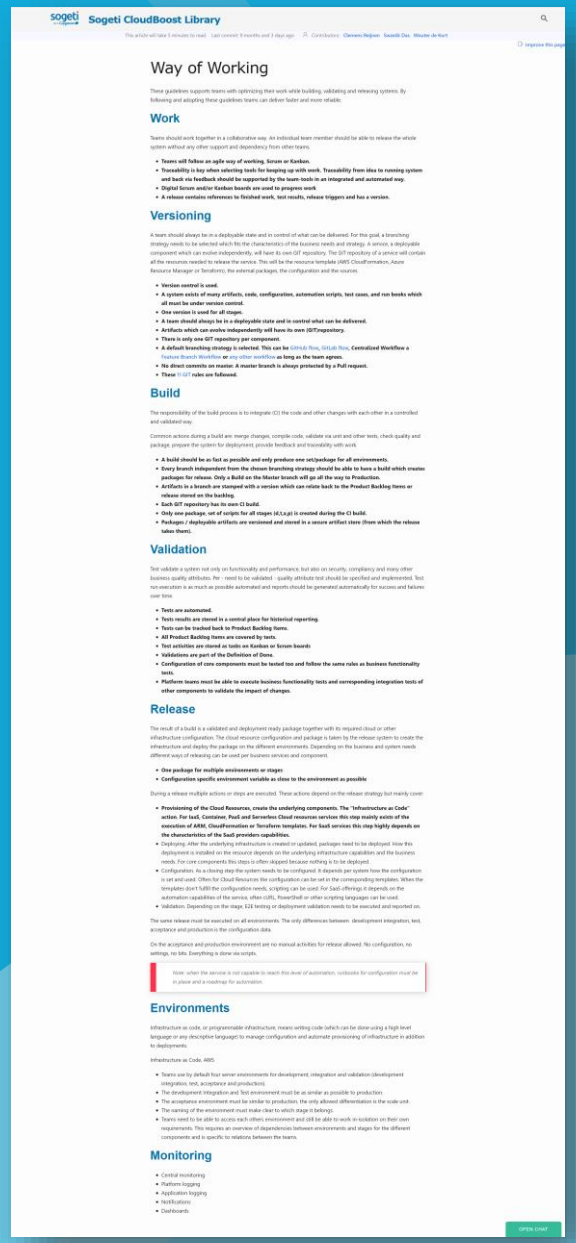
DevOps GOALS

reduced cycle time.
early feedback.
increased predictability.
lower software delivery risks.

DevOps Principles

High quality by default.
All versioned.
No manual activities.
Continuous improvement.
Everyone is/ feels responsible.
Transparency everywhere.

DevOps Practices



Sogeti CloudBoost Library

Way of Working

These guidelines supports teams with optimizing their work while building, validating and releasing systems. By following and adopting these guidelines teams can deliver faster and more reliable.

Work

Teams should work together in a collaborative way. An individual team member should be able to release the whole system without any other support and dependency from other teams.

- **Teams will follow an agile way of working, Scrum or Kanban.**
- **Traceability is key when selecting tools for keeping up with work. Traceability from idea to running system and back via feedback should be supported by the team-tools in an integrated and automated way.**
- **Digital Scrum and/or Kanban boards are used to progress work**
- **A release contains references to finished work, test results, release triggers and has a version.**

Versioning

A team should always be in a deployable state and in control of what can be delivered. For this goal, a branching strategy needs to be selected which fits the characteristics of the business needs and strategy. A service, a deployable component which can evolve independently, will have its own GIT repository. The GIT-repository of a service will contain all the resources needed to release the service. This will be the resource template (AWS CloudFormation, Azure Resource Manager or Terraform), the external packages, the configuration and the sources.

- **Version control is used.**
- **A system exists of many artifacts, code, configuration, automation scripts, test cases, and run books which all must be under version control.**
- **One version is used for all stages.**
- **A team should always be in a deployable state and in control what can be delivered.**
- **Artifacts which can evolve independently will have its own (GIT)repository.**
- **There is only one GIT repository per component.**
- **A default branching strategy is selected. This can be GitHub flow, GitLab flow, Centralized Workflow a Feature Branch Workflow or any other workflow as long as the team agrees.**
- **No direct commits on master. A master branch is always protected by a Pull request.**
- **These 9 GIT rules are followed.**

Build

The responsibility of the build process is to integrate (CI) the code and other changes with each other in a controlled and validated way.

Common actions during a build are: merge changes, compile code, validate via unit and other tests, check quality and package, prepare the system for deployment, provide feedback and traceability with work.

- **A build should be as fast as possible and only produce one set/package for all environments.**
- **Every branch independent from the chosen branching strategy should be able to have a build which creates packages for release. Only a Build on the Master branch will go all the way to Production.**
- **Artifacts in a branch are stamped with a version which can relate back to the Product Backlog Items or release stored on the backlog.**
- **Each GIT repository has its own CI build.**
- **Only one package, set of scripts for all stages (d,t,a,p) is created during the CI build.**
- **Packages / deployable artifacts are versioned and stored in a secure artifact store (from which the release takes them).**

Validation

Test validate a system not only on functionality and performance, but also on security, compliancy and many other business quality attributes. Per - need to be validated - quality attribute test should be specified and implemented. Test run execution is as much as possible automated and reports should be generated automatically for success and failures over time.

- **Tests are automated.**
- **Tests results are stored in a central place for historical reporting.**
- **Tests can be tracked back to Product Backlog Items.**
- **All Product Backlog Items are covered by tests.**
- **Test activities are stored as tasks on Kanban or Scrum boards**
- **Validations are part of the Definition of Done.**
- **Configuration of core components must be tested too and follow the same rules as business functionality tests.**
- **Platform teams must be able to execute business functionality tests and corresponding integration tests of other components to validate the impact of changes.**

Release

The result of a build is a validated and deployment ready package together with its required cloud or other infrastructure configuration. The cloud resource configuration and package is taken by the release system to create the infrastructure and deploy the package on the different environments. Depending on the business and system needs different ways of releasing can be used per business services and component.

- **One package for multiple environments or stages**
- **Configuration specific environment variable as close to the environment as possible**

During a release multiple actions or steps are executed. These actions depend on the release strategy but mainly cover:

- **Provisioning of the Cloud Resources, create the underlying components. The "Infrastructure as Code" artifacts. For IaaS, Container, PaaS and Serverless Cloud resources services this step mainly exists of the execution of ARM, CloudFormation or Terraform templates. For SaaS services this step highly depends on the characteristics of the SaaS providers capabilities.**
- Deploying. After the underlying infrastructure is created or updated, packages need to be deployed. How this deployment is installed on the resource depends on the underlying infrastructure capabilities and the business needs. For core components this steps is often skipped because nothing is to be deployed.
- Configuration. As a closing step the system needs to be configured. It depends per system how the configuration is set and used. Often for Cloud Resources the configuration can be set in the corresponding templates. When the templates don't fulfill the configuration needs, scripting can be used. For SaaS offerings it depends on the automation capabilities of the service, often cURL, PowerShell or other scripting languages can be used.
- Validation. Depending on the stage, E2E testing or deployment validation needs to be executed and reported on.

The same release must be executed on all environments. The only differences between development integration, test, acceptance and production is the configuration data.

On the acceptance and production environment are no manual activities for release allowed. No configuration, no settings, no bits. Everything is done via scripts.

> Note: when the service is not capable to reach this level of automation, runbooks for configuration must be in place and a roadmap for automation.

Environments

Infrastructure as code, or programmable infrastructure, means writing code (which can be done using a high level language or any descriptive language) to manage configuration and automate provisioning of infrastructure in addition to deployments.

Infrastructure as Code, AWS

- Teams use by default four server environments for development, integration and validation (development integration, test, acceptance and production).
- The development, integration and Test environment must be as similar as possible to production.
- The acceptance environment must be similar to production, the only allowed differentiation is the scale unit.
- The naming of the environment must make clear to which stage it belongs.
- Teams need to be able to access each others environment and still be able to work in isolation on their own requirements. This requires an overview of dependencies between environments and stages for the different components and is specific to relations between the teams.

Monitoring

- Central monitoring
- Platform logging
- Application logging
- Notifications
- Dashboards

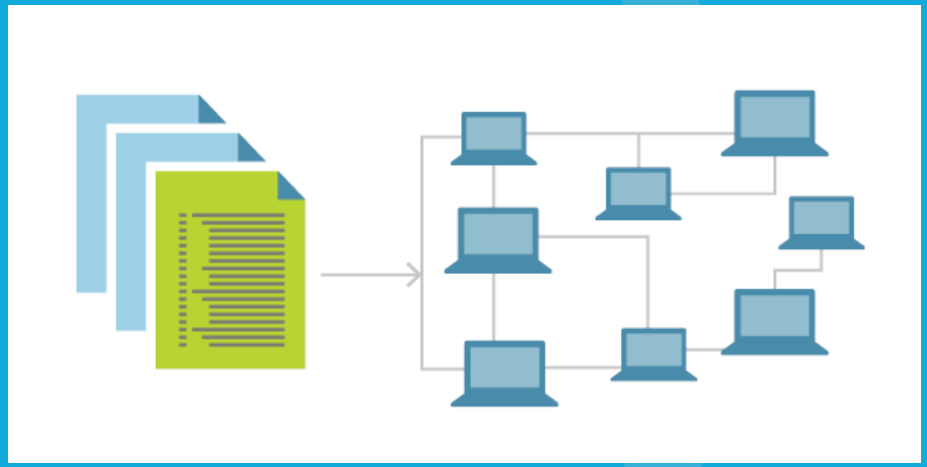
As Code paradigm

The ‘as code’ paradigm is about being able to reproduce and/or restore a full environment within minutes based on recipes and automation, managed as code.

https://martinfowler.com/bliki/InfrastructureAsCode.html

IaC is a key DevOps practice

Infrastructure as Code is the management of infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, using the same versioning as DevOps team uses for source code. Like the principle that the same source code generates the same binary, an IaC model generates the same environment every time it is applied. IaC is a key DevOps practice and is used in conjunction with continuous delivery.



https://docs.microsoft.com/nl-nl/azure/devops/learn/what-is-infrastructure-as-code

Everything As Code

Infrastructure as Code

Configuration as Code

Pipelines as Code

Documentation as Code

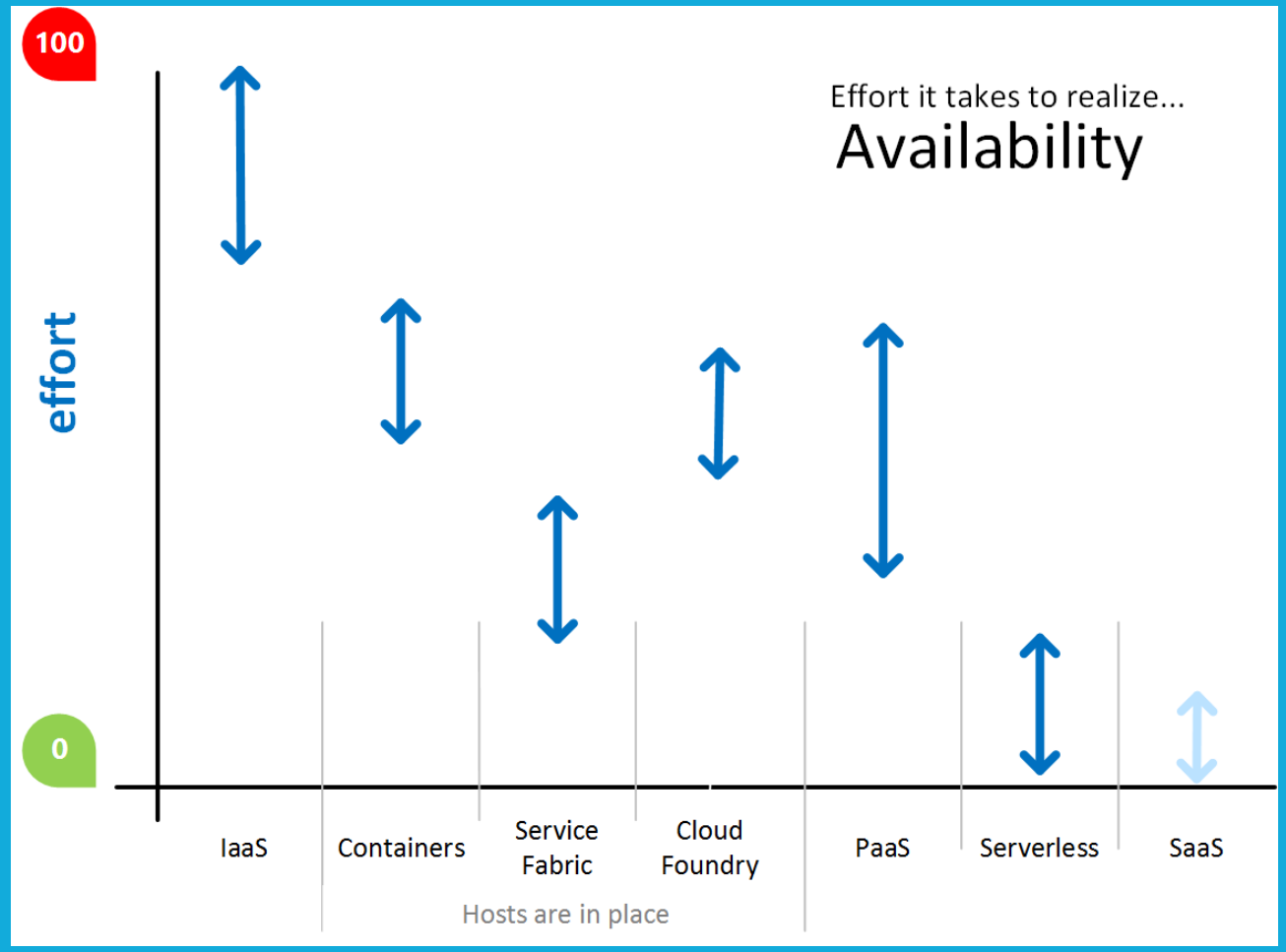
Tests as Code

Security policy as code

Compliance and governance as code

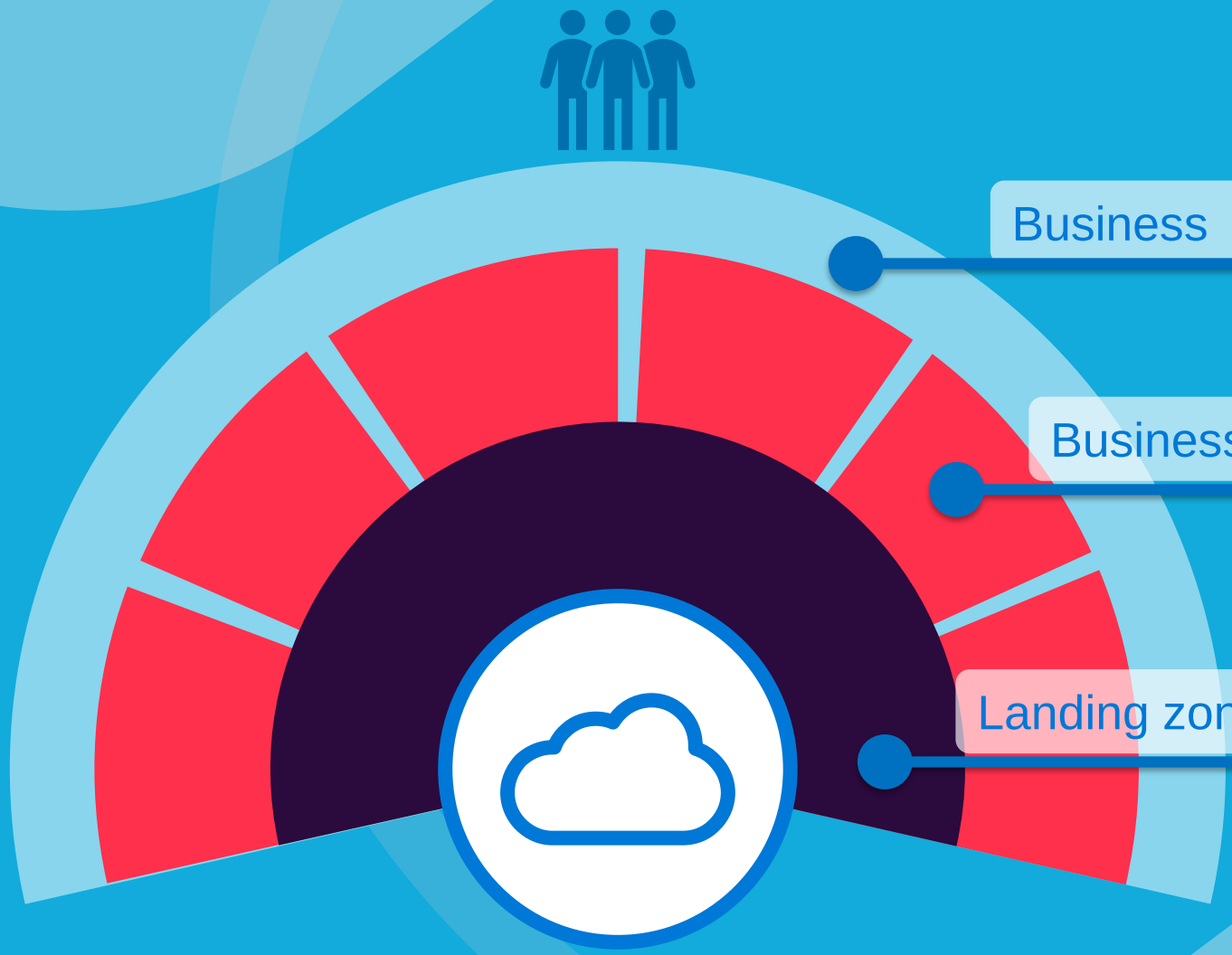
Platform

The Cloud tradeoff from a **business quality attribute** perspective.



https://www.linkedin.com/pulse/cloud-tradeoff-from-business-quality-attribute-clemens-reijnen/

Platform



Business Processes

Business Systems

Landing zone

Customer Capabilities

Process Capabilities

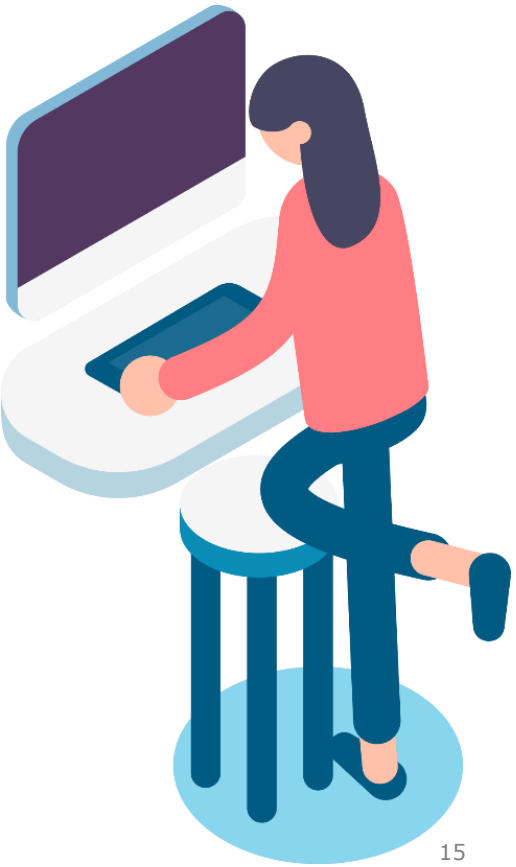
Business Capabilities

Company Capabilities

Cloud Capabilities

What slows your team down?

Stop trying to turn all your developers into DevOps pros, rather speed up on business functionality…

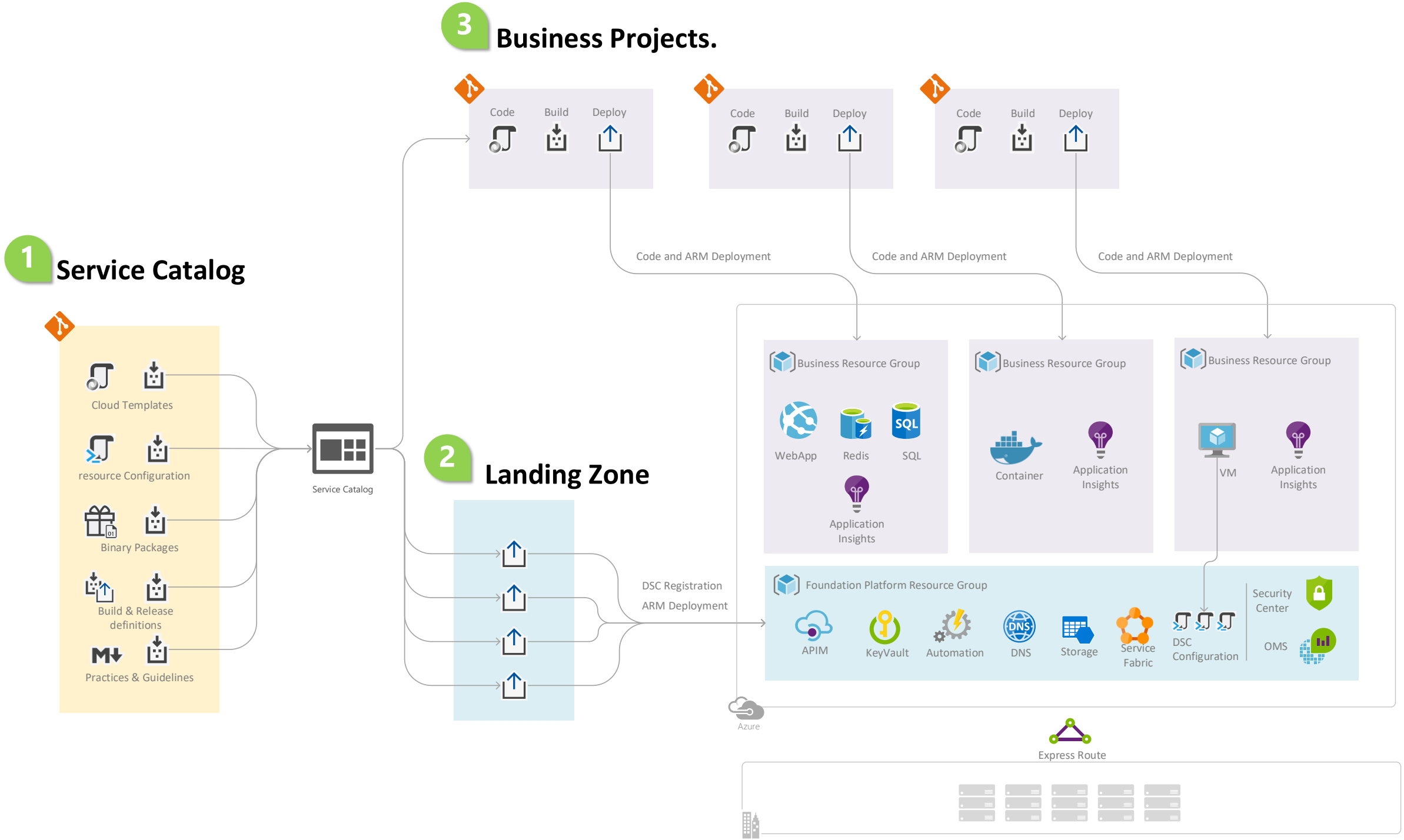


Organizations

Consistency at scale

Maximum Cloud benefits

Enterprise DevOps challenges


3
Business Projects.
Code
Build
Deploy
Code
Build
Deploy
Code
Build
Deploy
1
Service Catalog
Code and ARM Deployment
Code and ARM Deployment
Code and ARM Deployment
Business Resource Group
Business Resource Group
Business Resource Group
Cloud Templates
resource Configuration
Binary Packages
Build & Release definitions
Practices & Guidelines
Service Catalog
2
Landing Zone
WebApp
Redis
SQL
Application Insights
Container
Application Insights
VM
Application Insights
DSC Registration
ARM Deployment
Foundation Platform Resource Group
APIM
KeyVault
Automation
DNS
Storage
Service Fabric
DSC Configuration
Security Center
OMS
Azure
Express Route

Inner source

People and management

Product

Practices and Tools

Sogeti **CloudBoost** library

Inner source, developing **open source** software within **organizations**