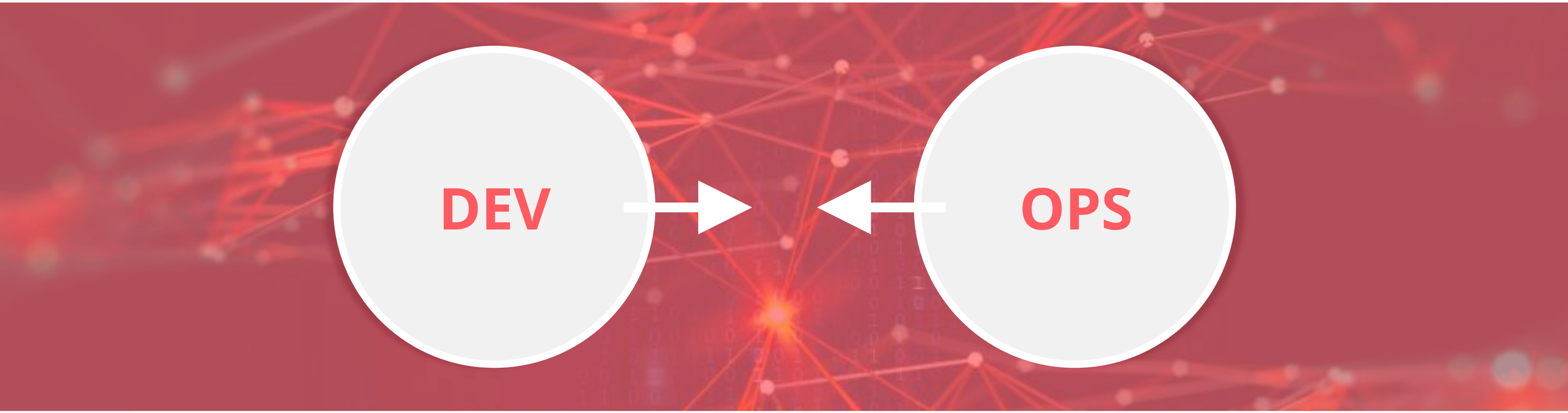
DevOps Transformation

Learnings & best practices for enterprise DevOps transformation

QBurst

DEVOPS TRANSFORMATION

INTRODUCTION



- An approach based on agile and lean principles
- Provides a path to faster delivery of software without compromising on reliability or security
- Improves delivery flow, reduces lead time, and improves acceptance ratio during handoffs
- Enables cross-functional collaboration within value streams

DEVOPS TRANSFORMATION

WHY DEVOPS?

Iterative

Processes that enable iterative delivery, built-in reviews and in-process testing

Collaboration

Enhanced collaboration and integration between developers, QA, and IT Operations

Automation

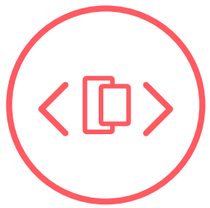
Tools to automate activities such as development, testing, and deployment

Continuous Integration

Frequent merging of code into a shared repository to detect problems early

Quantification

Gauge improvement of processes by tracking KPIs such as deployment frequency, failures, and lead time



Version Control

Applying version code not just to source code, but to configurations and environment

Holistic

Holistic approach to agile — covers all processes related to development, testing, and deployment

CULTURAL TRANSFORMATION

CULTURAL TRANSFORMATION

PREREQUISITES FOR DEVOPS TRANSFORMATION

Organizational culture and strategy

- Readiness for change, buy-in from senior management, focus on innovation, coordination between teams, existing skill-sets and ramp-up challenges, customer focus

Organizational structure

- Willingness to standardize and break down silos that exist within processes and line of businesses

CULTURAL TRANSFORMATION

EVALUATING THE ORGANIZATIONAL ARCHETYPES

FUNCTIONAL

- Hierarchically divided into smaller groups with specific tasks or roles
- Optimized for skills, labor division, and cost-reduction

MARKET-ORIENTED

- Flat organization with resources fully dedicated to common project activities
- Optimize for delivering value to customers

MATRIX

- Hybrid organizational structure with functional divisions
- Attempts to combine both Functional and Market orientations

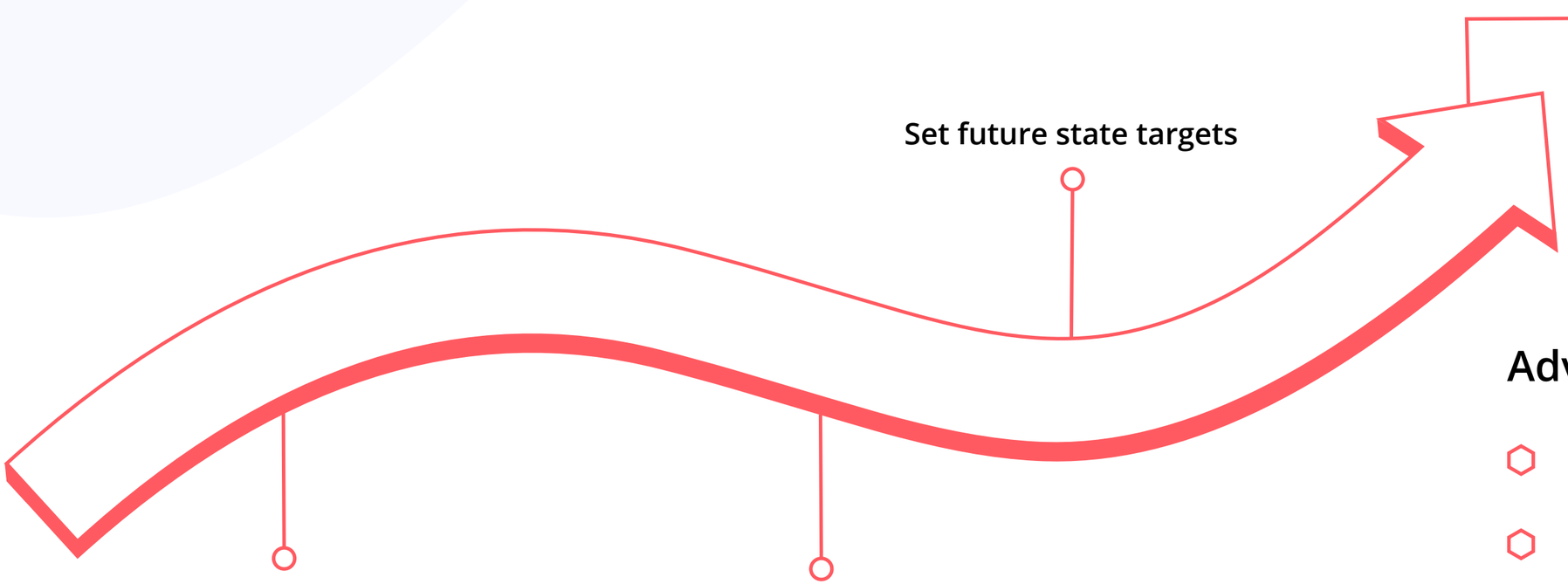
Your existing organizational structure may work with a few change

- Move toward market-orientation by embedding functional engineers (QA, Infosec, Ops) into each service team
- Restructure service delivery cycle with DevOps automation in mind
- Embrace a culture of communication and collaboration throughout the organization
- Identify opportunities where automation can bring about efficiency and productivity

PROCESS TRANSFORMATION

PROCESS TRANSFORMATION

VALUE STREAM MAPPING



Advantages

- Identify bottlenecks
- Improve visibility and traceability
- Eliminate redundant and wasteful processes
- Identify opportunities for automation
- Bring process clarity with data and visuals
- Highlight results and KPIs

DEVOPS TRANSFORMATION: PHASED APPROACH

ASSESS

- Assess current state of Dev and Ops
- Gap analysis - Identify opportunities for automation
- Identify challenges in build, deployment and release management

IMPLEMENT

- Standardize governance structure and processes
- Automate build, release, and deployment
- Establish program governance -communication plan, set KPIs
- Initiate pilots

MAINTAIN

- Monitor adherence to SLAs and KPIs
- Implement continuous improvement plans and governance models

TECHNOLOGY TRANSFORMATION

TECHNOLOGY TRANSFORMATION

SOFTWARE ARCHITECTURE - MOVING FROM REACTIVE TO SCALABLE

AGILE

DEVOPS

ARCHITECTURE

N-tiered Decoupled

↓

Microservices

Architecture should be sufficiently decoupled, so work can be done without excessive coordination and communication — **Microservices architecture is ideal for decoupling**

PACKAGING

VM

↓

Containers

Transition to containers — lightweight compared to VMs

Launch multiple containers in minutes

INFRASTRUCTURE

Hosted

↓

Cloud

Flexible cloud services to increase or decrease infrastructure based on requirement
Reduced CAPEX planning; pricing based on use

TECHNOLOGY TRANSFORMATION

VERSION CONTROL

WHY?

- Standardize coding practices
- Keep a log and view changes
- Enables team to carry out development in parallel
- Faster debugging of deployment failures and production issues
- Infrastructure as Code: Anyone can create an environment using information from source control

WHAT?

- Code as well as infrastructure should be version controlled
- Operations-related artifacts
- Source code, automation scripts, deployment scripts, installation environment definition, infrastructure configuration and documents

HOW?

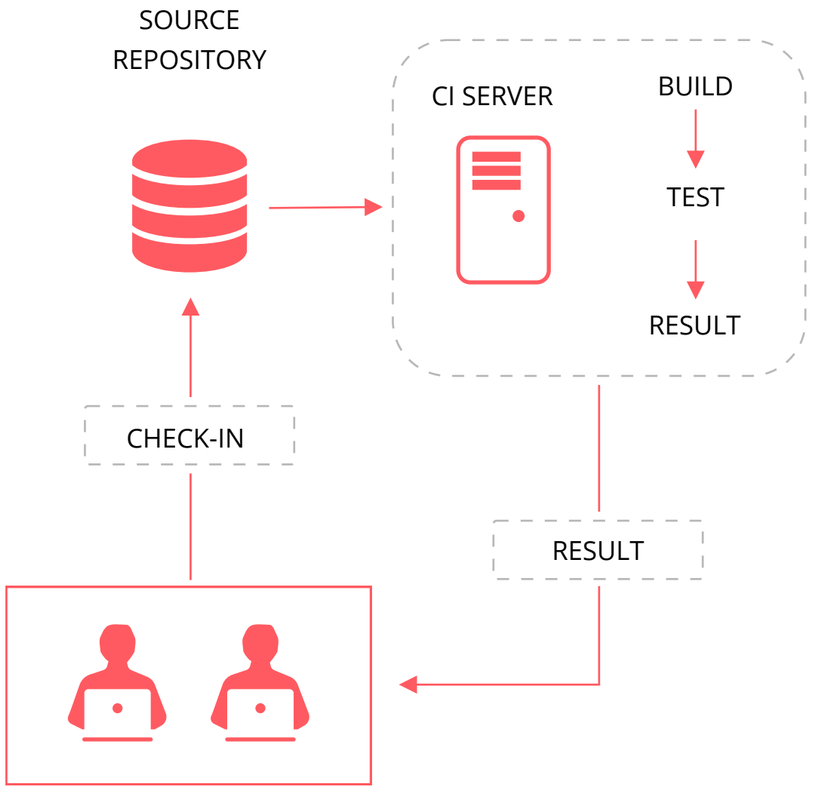
- Version controlling tools
  - View difference between versions
  - Single source of truth

TECHNOLOGY TRANSFORMATION

CONTINUOUS INTEGRATION

Integrating becomes exponentially difficult with an increase in branches or the number of changes in each branch.

- Merging into trunk should be part of everyone’s daily work
- Create a comprehensive automated test suite: Automated tests to be written for new features, enhancements, and bug fixes
- Integrate in smaller batches
- Run locally before committing to CI server

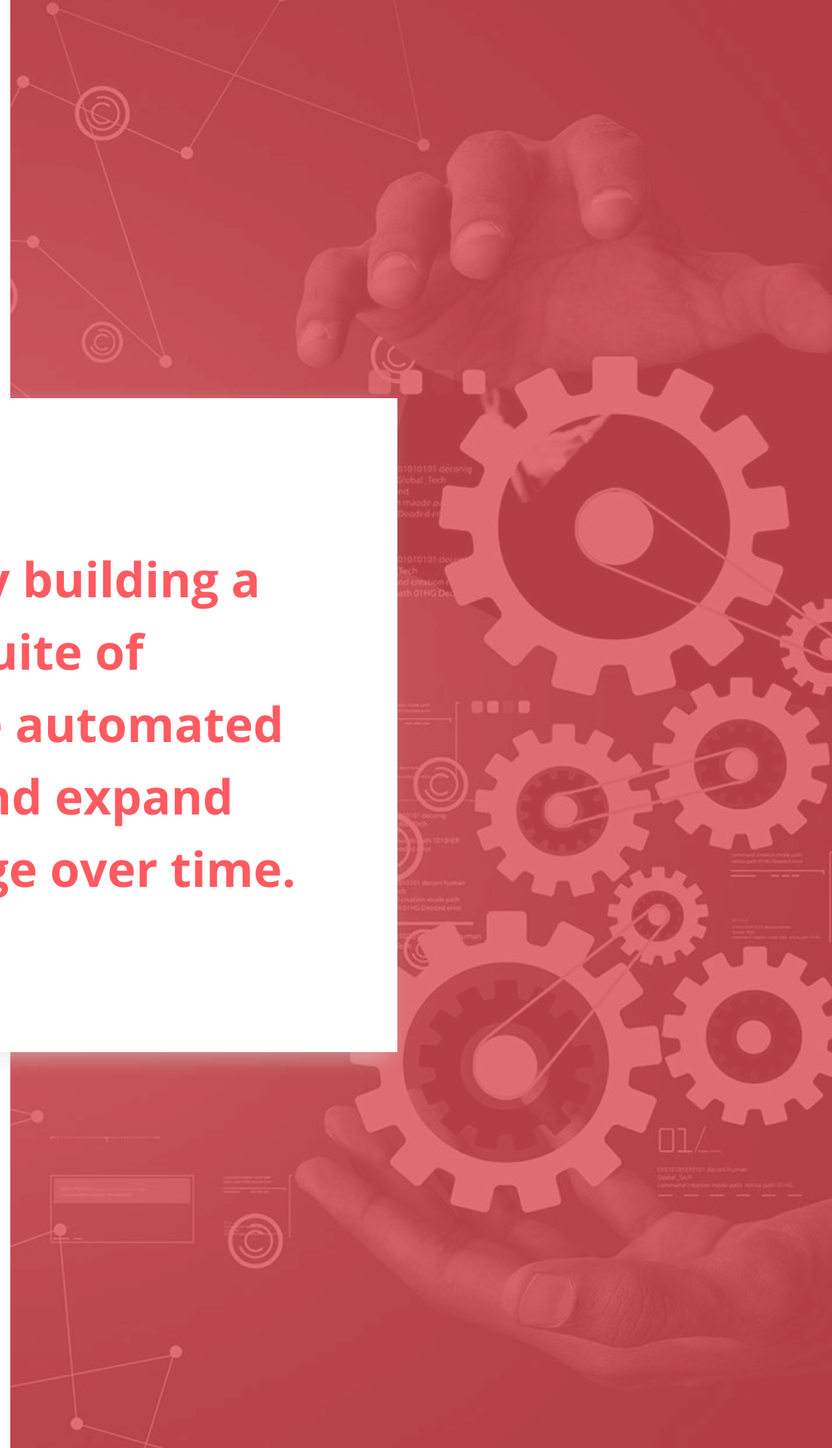


DEVOPS TRANSFORMATION

AUTOMATED TESTING - BEST PRACTICES

- Code checked into version control must be automatically built and tested in a production-like environment
- Unit tests, acceptance tests, integration tests can be automated
- Unit and acceptance tests should run quickly — running them in parallel is recommended
- Automated testing on developer workstation can provide faster feedback
- Non-functional tests such as performance testing and security testing should be automated
- Automated tests should be reliable — false positives and unreliable tests create more problems than they solve
- A small number of reliable tests is better than a large number of unreliable tests

Start by building a small suite of reliable automated tests and expand coverage over time.



TECHNOLOGY TRANSFORMATION

DEPLOYMENT PIPELINE

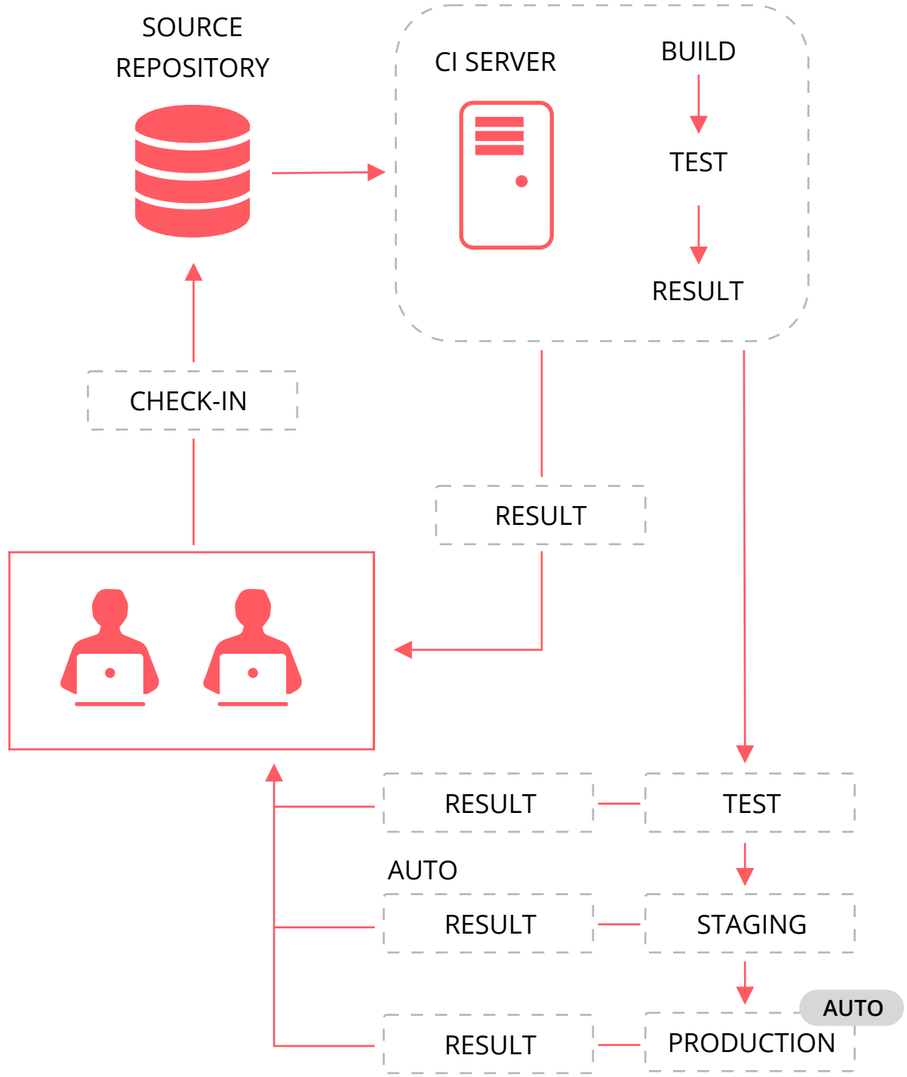
- Utilize visualization tools to monitor pipeline and ensure green-build state
- Create a virtual Andon Cord
  - If a change that causes build to fail is introduced, no new work should be accepted until the problem is fixed
  - Notify testers and developers when a problem needs to be fixed
- Static code analysis, duplication/test coverage analysis, and checking style
- Ensure staging environment is identical to production environment
- Some tools can be run from IDE or during pre-commit (via pre-commit hooks) to enable faster feedback
- Create containers as part of the build process

TECHNOLOGY TRANSFORMATION

CONTINUOUS INTEGRATION

Developers should be able to deploy on-demand, enable multiple deploys per day.

- Keep code in a deployable state at all times
- Seamless feedback loop between users and developers
- Smaller sprints ensure faster turnaround time for bug fixes
- Select and share the right tools and procedures between teams
- Identify bottlenecks in deployment process and streamline over time



DEPLOYMENT STRATEGIES: INFRASTRUCTURE-BASED

BLUE-GREEN DEPLOYMENT

- Deploy complete application components, services twice
- Old version in blue and new version in green side by side
- To cut over to the new version and roll back to old, change the load balancer or router setting

CANARY RELEASE

- Incrementally upgrade to the latest version
- First build can go to employees, then to a larger group, and finally to everyone
- If a problem is discovered in an early stage, build goes no further

CLUSTER IMMUNE SYSTEM

- Monitors critical system metrics when a new version is rolled out in a canary release.
- Automatically rolls back the deployment in case of high stats

DEPLOYMENT STRATEGIES: APPLICATION-BASED

FEATURE TOGGLES

- Control who can access a new feature to test a feature on production with a select group of users.
- Implement a toggle router to dynamically control which code path is live
- Toggle Router can make decisions based on environment-specific configurations
- Helps release near bug-free features

DARK LAUNCHES

- Releasing production-ready features to a subset of users first
- Helps get real user feedback, test for bugs, and assess infrastructure performance
- In case something goes wrong during deployment, it’s easy to roll back the changes and fix the problem with the old infrastructure/code still in place

DEVOPS TRANSFORMATION

ENVIRONMENT: DOS & DON'TS

Developers should have **the ability to create production-like environments on-demand.**

Automate environment creation process. This applies to development, testing, and production environments.

Utilize tools such as chef to **automatically configure newly provisioned environments.**

Make infrastructure easier to build than repair.

Entire application stack and environment can be bundled into containers. **Package applications into deployable containers.**

Verify that the application runs as expected in a production-like environment before the end of a sprint.

Inconsistently constructed environments and not putting changes back to version control can create problems.

Immutable infrastructure - manual changes to production environment are not allowed.

DEVOPS TRANSFORMATION

MONITORING

- Have an integrated monitoring system for Dev and Ops
- Move all monitored data to a central location
- Derive metrics from monitored data
  - Plot metrics as graphs
  - Display deployment events on the same graph to correlate problems with deployments
  - Statistically analyze collected metrics to identify deviations

    Example: Generate an alert if the metric (e.g. page load time) is 3 standard-deviations away from the mean (this assumes the metric has a Gaussian distribution)

    Another strategy is outlier identification
    Example: In a cluster of thousands of nodes, if the performance metrics of a node deviates from the normal, it can be taken down
- Alerts: Generate alerts only for indicators that predict outages. Too many alerts could cause alert fatigue.
- Anomaly detection: Use statistical tools (Excel, SPSS, SAS, R etc.) on datasets to find anomalies.
  Example: If orders fall below 50% of the normal (expected number of orders based on historical trend) on Wednesday morning
- Kolmogorov-Smirnov test: Find similarities/differences in seasonal/periodic data

WHAT SHOULD BE MONITORED?

End-to-end monitoring of the entire software stack is essential.

- Applications
- Databases
- Servers and networks
- Builds
- Automated tests
- Deployments
- All environments
  (development, testing, staging, production)

Tools

Grafana GRAPHITE Nagios®

ADOPTION OF TOOLS

DEVOPS TRANSFORMATION

TOOLS

SCM

git
SUBVERSION
GitLab
Bitbucket

CONTAINERIZATION

docker
kubernetes
Amazon ECS
VAGRANT

CI/CD

AWS Developer Tools
Bamboo
Jenkins
Capistrano

LOG

graylog
loggly
elastic
fluentd

INFRA & CONFIG MANAGEMENT

ANSIBLE
puppet labs
HashiCorp Terraform

COLLABORATION

slack
JIRA
Trello

TEST

appium
Selenium
APACHE JMeter™

SECURITY

Fail2Ban
TREND MICRO

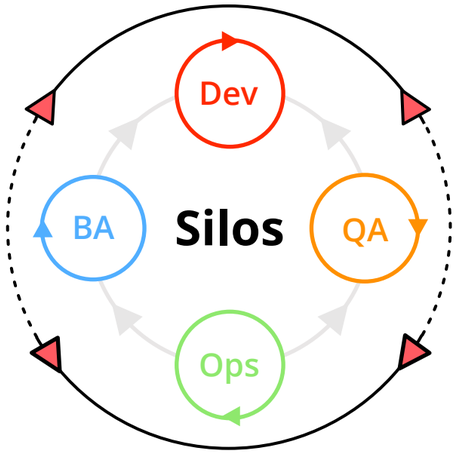
ICINGA
Prometheus
New Relic.
Grafana
SENTRY
PRTG NETWORK MONITOR

TRANSITION

DEVOPS TRANSITION

PATH TO PROGRESSIVE AGILITY

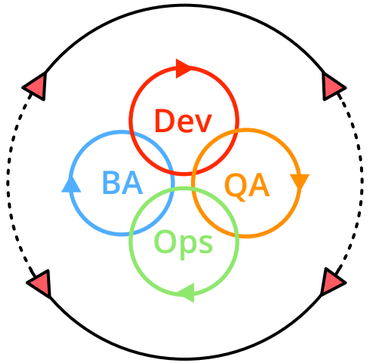
MATURITY LEVEL 0



No DevOps

Teams work in silos, manual deployment, longer time to market, outages

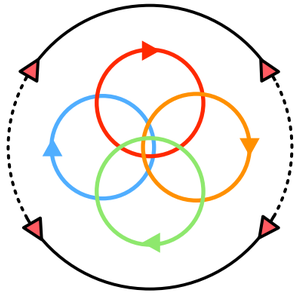
MATURITY LEVEL 1



Fundamental Devops

Automated build, crossfunctional teams, focus on product, shared objectives

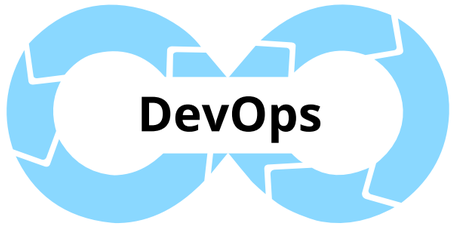
MATURITY LEVEL 2



Managed Devops

Consistent and repeatable processes, Integrated tool chain, automated tests and deployment - continuous delivery

MATURITY LEVEL 3

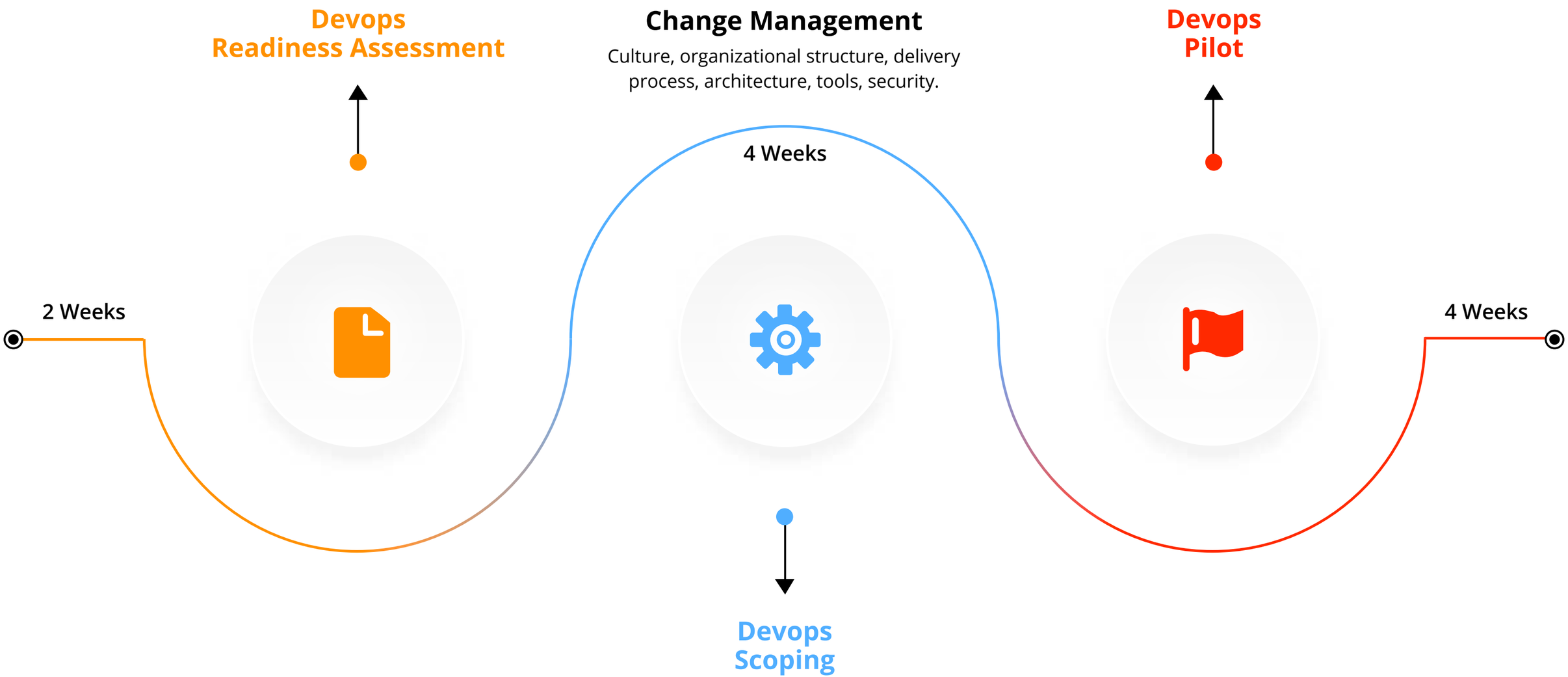


Optimizing DevOps

One team, Infrastructure as Code, zero downtime deployments, any time deployment

DEVOPS TRANSITION

HOW WE DO IT



QBurst

14150 Newbrook Drive
Suite 115
Chantilly, VA 20151, USA

www.qburst.com