DevOps

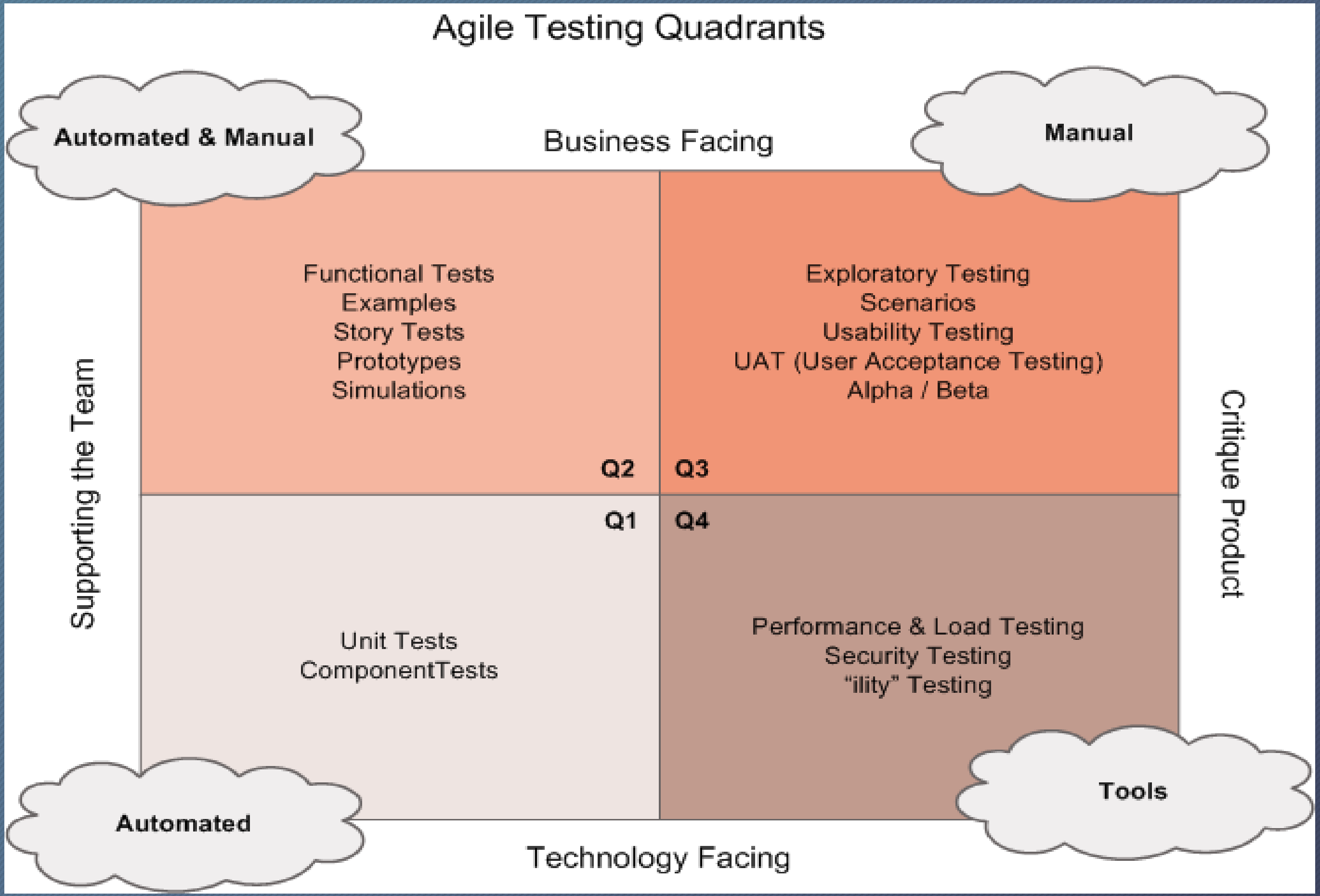
Week 12

Murtaza Munawar Fazal

Continuous Deployment - Testing

- Business facing - the tests are more functional and often executed by end users of the system or by specialized testers that know the problem domain well.
- Supporting the Team - it helps a development team get constant feedback on the product to find bugs fast and deliver a product with quality build-in.
- Technology facing - the tests are rather technical and non-meaningful to business people. They're typical tests written and executed by the developers in a development team.
- Critique Product - tests that are there to validate the workings of a product on its functional and non-functional requirements.

Continuous Deployment - Testing



Continuous Deployment - Testing

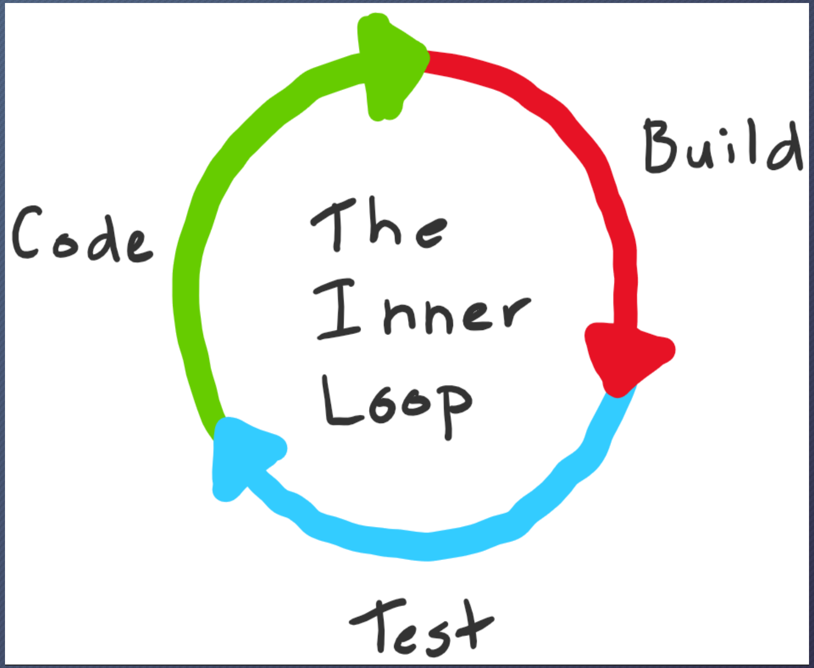
- We can put Unit tests, Component tests, and System or integration tests in the first quadrant.
- In quadrant two, we can place functional tests, Story tests, prototypes, and simulations. These tests are there to support the team in delivering the correct functionality and are business-facing since they're more functional.
- In quadrant three, we can place tests like exploratory, usability, acceptance, etc.
- We place performance, load, security, and other non-functional requirements tests in quadrant four.

Continuous Deployment - Testing

- Test that are easy to automate or automated by nature are in quadrants 1 and 4.
- Tests that are automatable but most of the time not automated by nature are the tests in quadrant 2. Tests that are the hardest to automate are in quadrant 3.
- Tests that can't be automated or are hard to automate are tests that can be executed in an earlier phase and not after release.
- We call shift-left, where we move the testing process towards the development cycle.
- We need to automate as many tests as possible and test them.
- A few of the principles we can use are:
  - Tests should be written at the lowest level possible.
  - Write once, run anywhere, including the production system.
  - The product is designed for testability.
  - Test code is product code; only reliable tests survive.
  - Test ownership follows product ownership.
- By testing at the lowest level possible, you'll find many tests that don't require infrastructure or applications to be deployed.

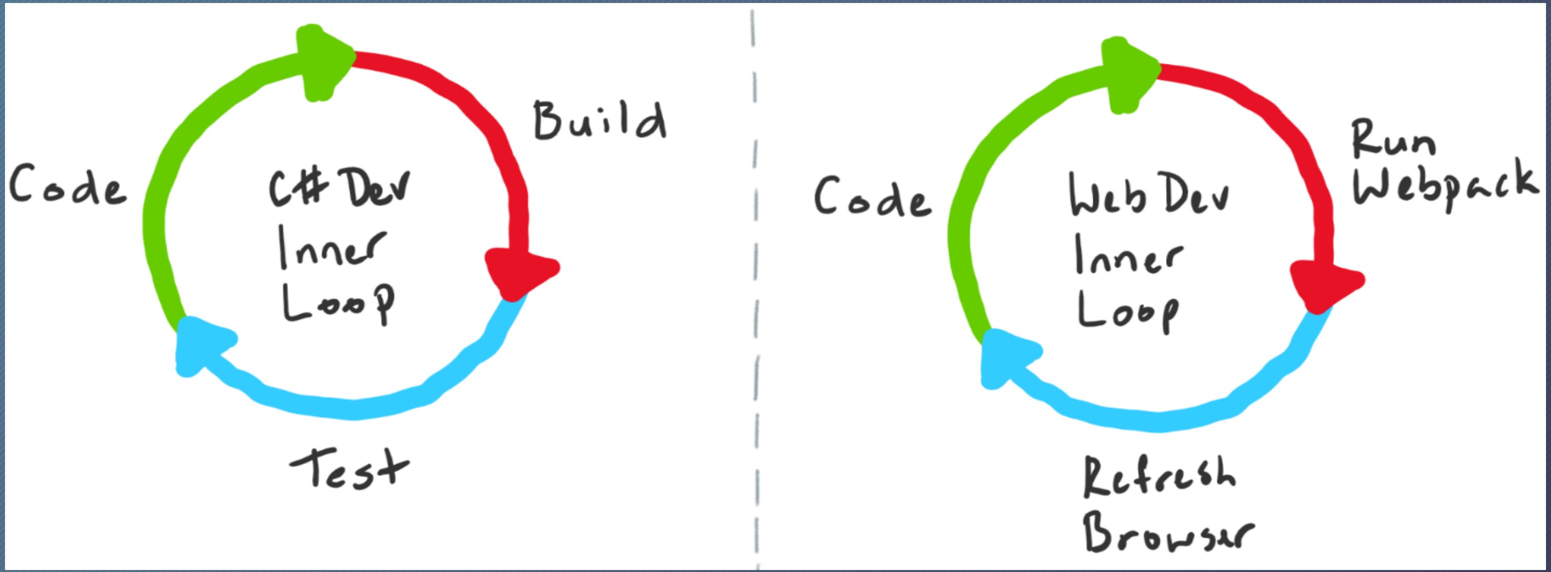
The Inner Loop

- The inner loop is the iterative process that a developer does before sharing their work with their team or the rest of the world.
- Developer's inner loop will depend significantly on the technologies they're working with, the tools being used, and their preferences.
- If working on a library, the inner loop would include coding, building, testing execution & debugging with regular commits to my local Git repository.

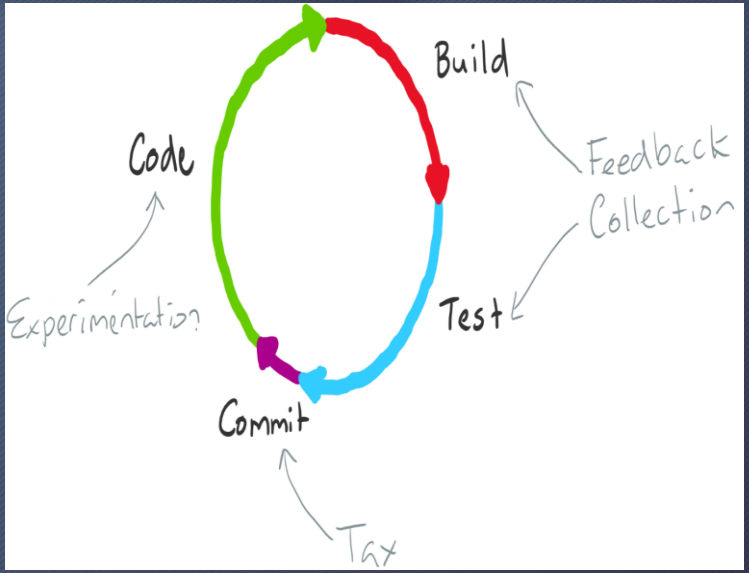


The Inner Loop

- On the other hand, if it was some web front-end work, the developer might be optimized around hacking on HTML & JavaScript, bundling, and refreshing the browser (followed by regular commits).
- Most codebases comprise multiple-moving parts, so the definition of a developer's inner loop on any single codebase might alternate depending on what is being worked on.



Understanding the Inner Loop



- The steps within the inner loop can be grouped into three broad buckets of activity - experimentation, feedback collection, and tax.
- If we flick back to the library development scenario, following 4 steps are:
  - Coding (Experimentation)
  - Building (Feedback Collection)
  - Testing / Debugging (Feedback Collection)
  - Committing (Tax)
- Of all the steps in the inner loop, coding is the only one that adds customer value.
- Building and testing code are essential, but ultimately, we use them to give the developer feedback about what they've written to see if it delivers sufficient value.
- Putting committing code in the tax bucket is perhaps a bit harsh, but the purpose of the bucket is to call out those activities that neither add value nor provide feedback.
- Tax is necessary to work. If it's unnecessary work, then it's a waste and should be eliminated.

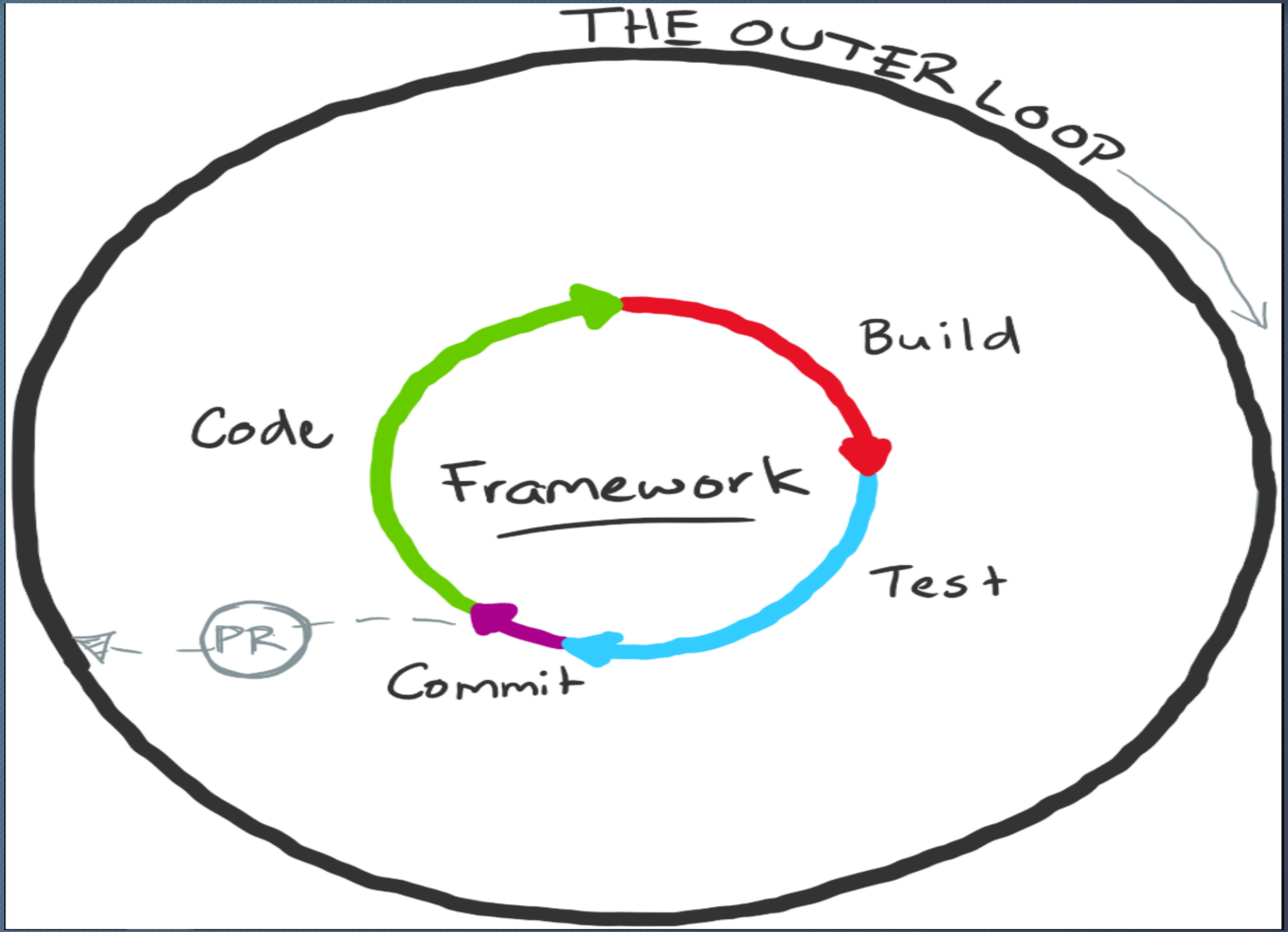
Understanding the Inner Loop

- Having categorized the steps within the loop, it's now possible to make some general statements:
  - You want to execute the loop as fast as possible and for the total loop execution time to be proportional to the changes made.
  - You want to minimize the time feedback collection takes but maximize the quality of the feedback that you get.
  - You want to minimize the tax you pay by eliminating it where it's unnecessary to run through the loop (can you defer some operations until you commit, for example).
  - As new code and more complexity are added to any codebase, the amount of outward pressure to increase the size of the inner loop also increases. More code means more tests, which means more execution time and slow execution of the inner loop.

Tangled Loops

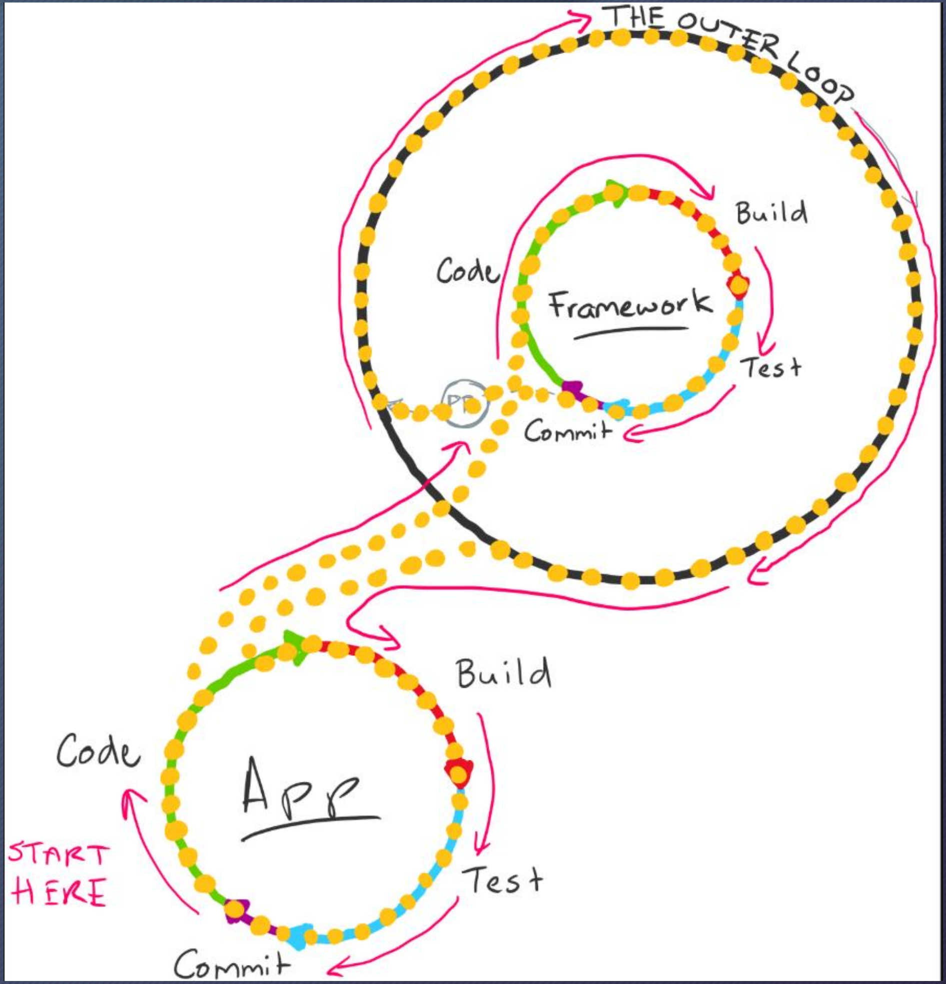
- Let us say that our monolithic codebase has an application-specific framework that does much heavy lifting.
- It would be tempting to extract that framework into a set of packages.
- To do this, you would pull that code into a separate repository (optional, but this is generally the way it's done), then set up a different CI/CD pipeline that builds and publishes the package.

Tangled Loops



Tangled Loops

- Initially, things might work out well. However, at some point in the future, you'll likely want to develop a new feature in the application that requires extensive new capabilities to be added to the framework.
- It's where teams that have broken up their codebases in suboptimal ways will start to feel pain.
- If you have to coevolve code in two separate repositories where a binary/library dependency is present, you'll experience some friction.
- Outer loops include tax, including code reviews, scanning passes, binary signing, release pipelines, and approvals.
- You don't want to pay that every time you've added a method to a class in the framework and now want to use it in your application.

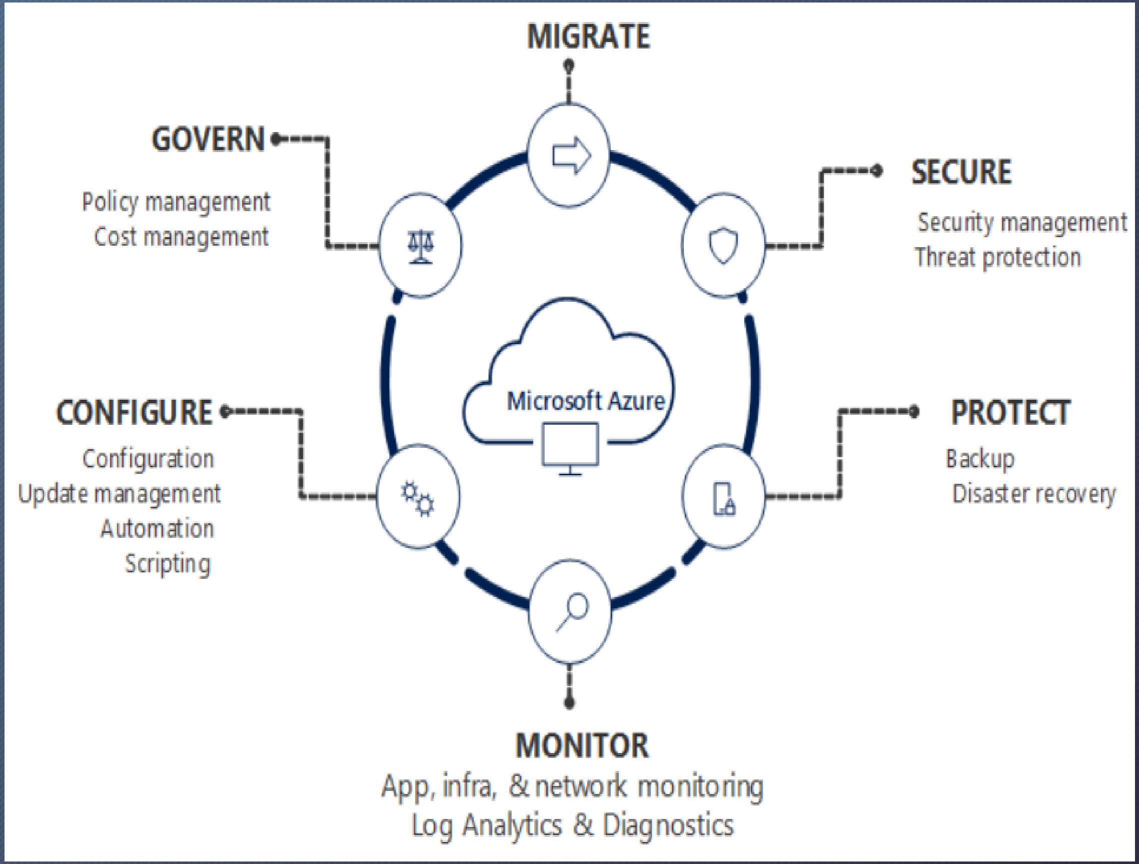


Continuous Monitoring

- Continuous monitoring refers to the process and technology required to incorporate monitoring across each phase of your DevOps and IT operations lifecycles.
- It helps to continuously ensure your application's health, performance, reliability, and infrastructure as it moves from development to production.
- Continuous monitoring builds on the concepts of Continuous Integration and Continuous Deployment (CI/CD), which help you develop and deliver software faster and more reliably to provide continuous value to your users.

Continuous Monitoring

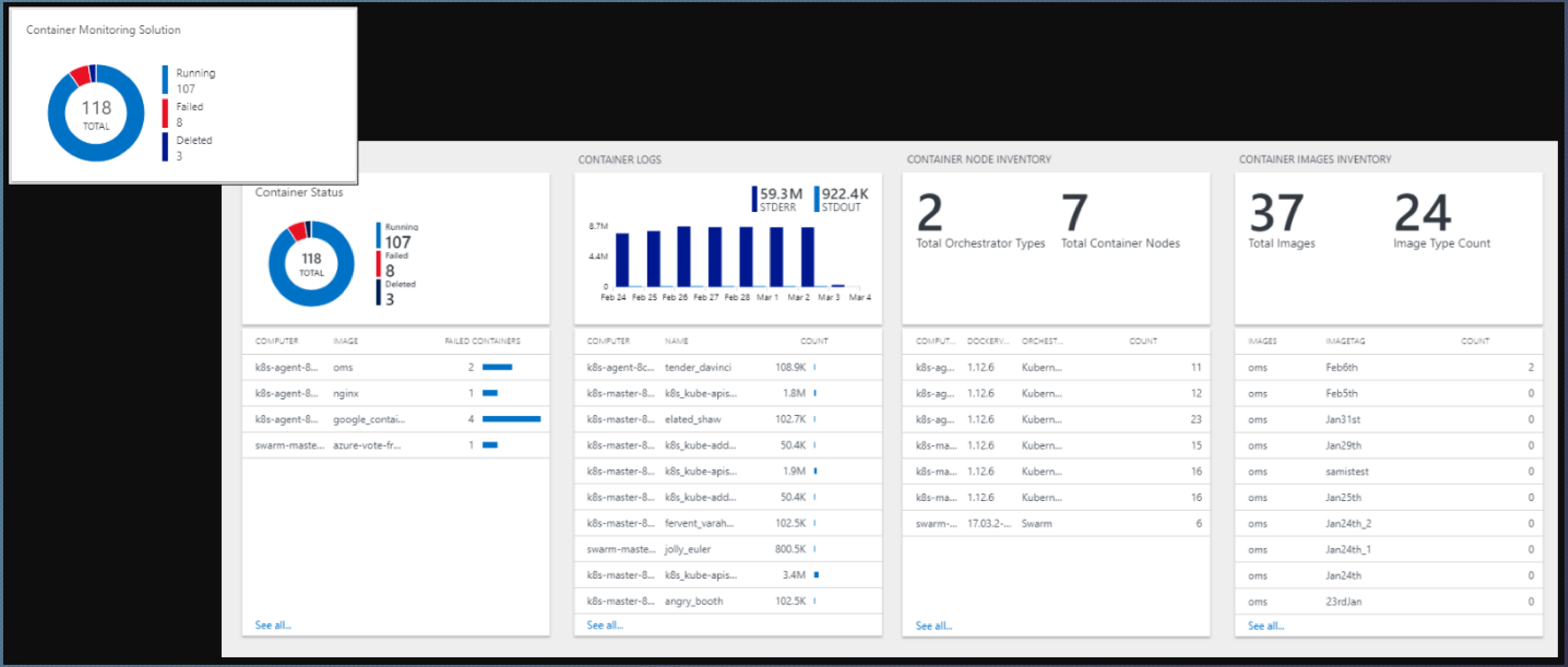
- Azure Monitor is the unified monitoring solution in Azure that provides full-stack observability across applications and infrastructure in the cloud and on-premises.
- It works seamlessly with Visual Studio and Visual Studio Code during development and testing and integrates with Azure DevOps for release management and work item management during deployment and operations.



Azure Monitor

- Enable monitoring for all your applications
- Enable monitoring for your entire infrastructure
- Combine resources in Azure Resource Groups
- Ensure quality through continuous deployment
- Create actionable alerts with actions

View designer in Azure Monitor



Container Monitoring Solution

118 TOTAL — Running 107, Failed 8, Deleted 3

CONTAINER LOGS

CONTAINER NODE INVENTORY

CONTAINER IMAGES INVENTORY

Container Status

118 TOTAL — Running 107, Failed 8, Deleted 3

COMPUTER	IMAGE	FAILED CONTAINERS
k8s-agent-8...	oms	2
k8s-agent-8...	nginx	1
k8s-agent-8...	google_contai...	4
swarm-maste...	azure-vote-fr...	1

See all...

59.3M STDERR | 922.4K STDOUT

COMPUTER	NAME	COUNT
k8s-agent-8c...	tender_davinci	108.9K
k8s-master-8...	k8s_kube-apis...	1.8M
k8s-master-8...	elated_shaw	102.7K
k8s-master-8...	k8s_kube-add...	50.4K
k8s-master-8...	k8s_kube-apis...	1.9M
k8s-master-8...	k8s_kube-add...	50.4K
k8s-master-8...	fervent_varah...	102.5K
swarm-maste...	jolly_euler	800.5K
k8s-master-8...	k8s_kube-apis...	3.4M
k8s-master-8...	angry_booth	102.5K

See all...

2 Total Orchestrator Types

7 Total Container Nodes

COMPUT...	DOCKERV...	ORCHEST...	COUNT
k8s-ag...	1.12.6	Kubern...	11
k8s-ag...	1.12.6	Kubern...	12
k8s-ag...	1.12.6	Kubern...	23
k8s-ma...	1.12.6	Kubern...	15
k8s-ma...	1.12.6	Kubern...	16
k8s-ma...	1.12.6	Kubern...	16
swarm-...	17.03.2-...	Swarm	6

See all...

37 Total Images

24 Image Type Count

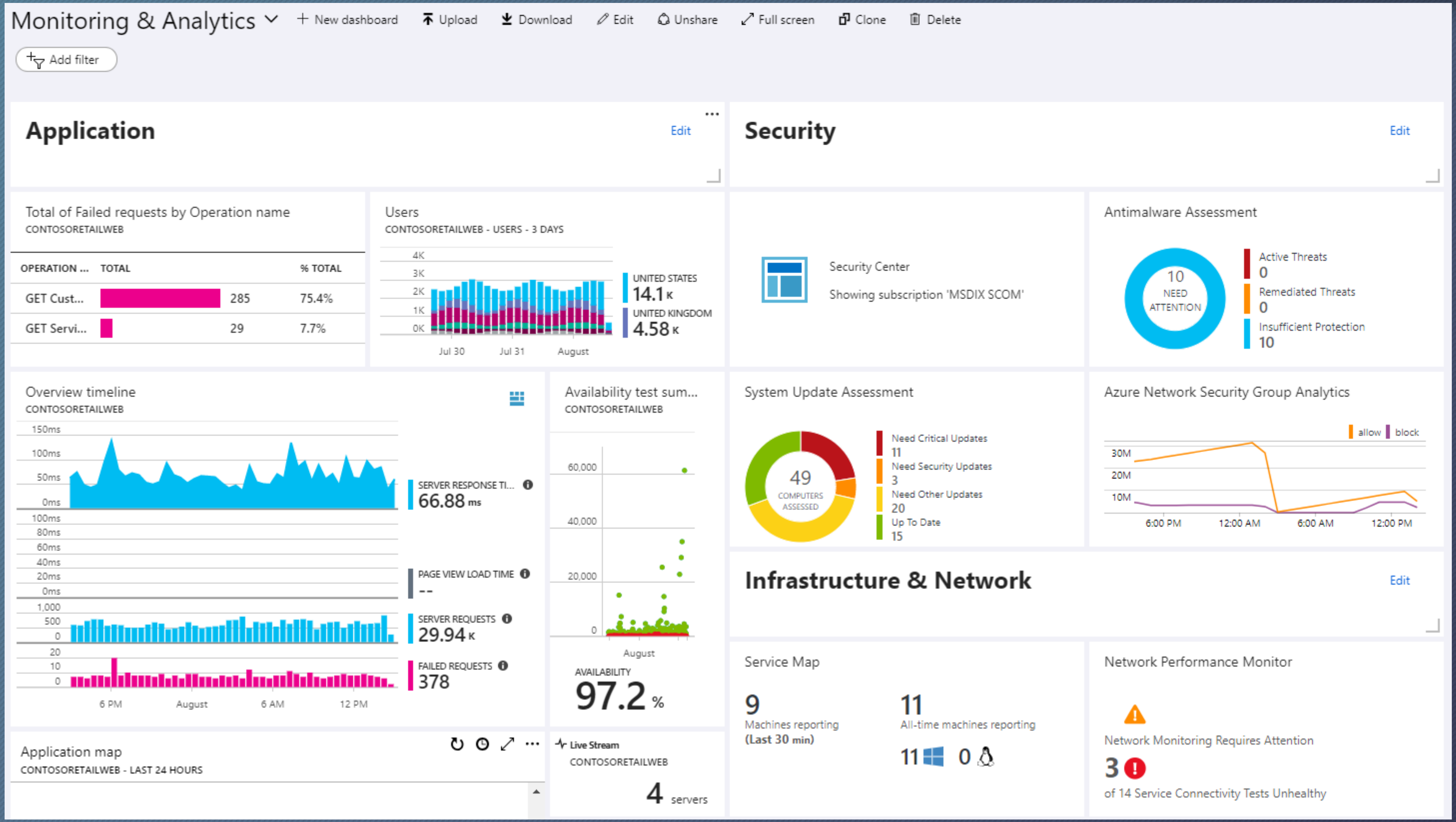
IMAGES	IMAGETAG	COUNT
oms	Feb6th	2
oms	Feb5th	0
oms	Jan31st	0
oms	Jan29th	0
oms	samistest	0
oms	Jan25th	0
oms	Jan24th_2	0
oms	Jan24th_1	0
oms	Jan24th	0
oms	23rdJan	0

See all...

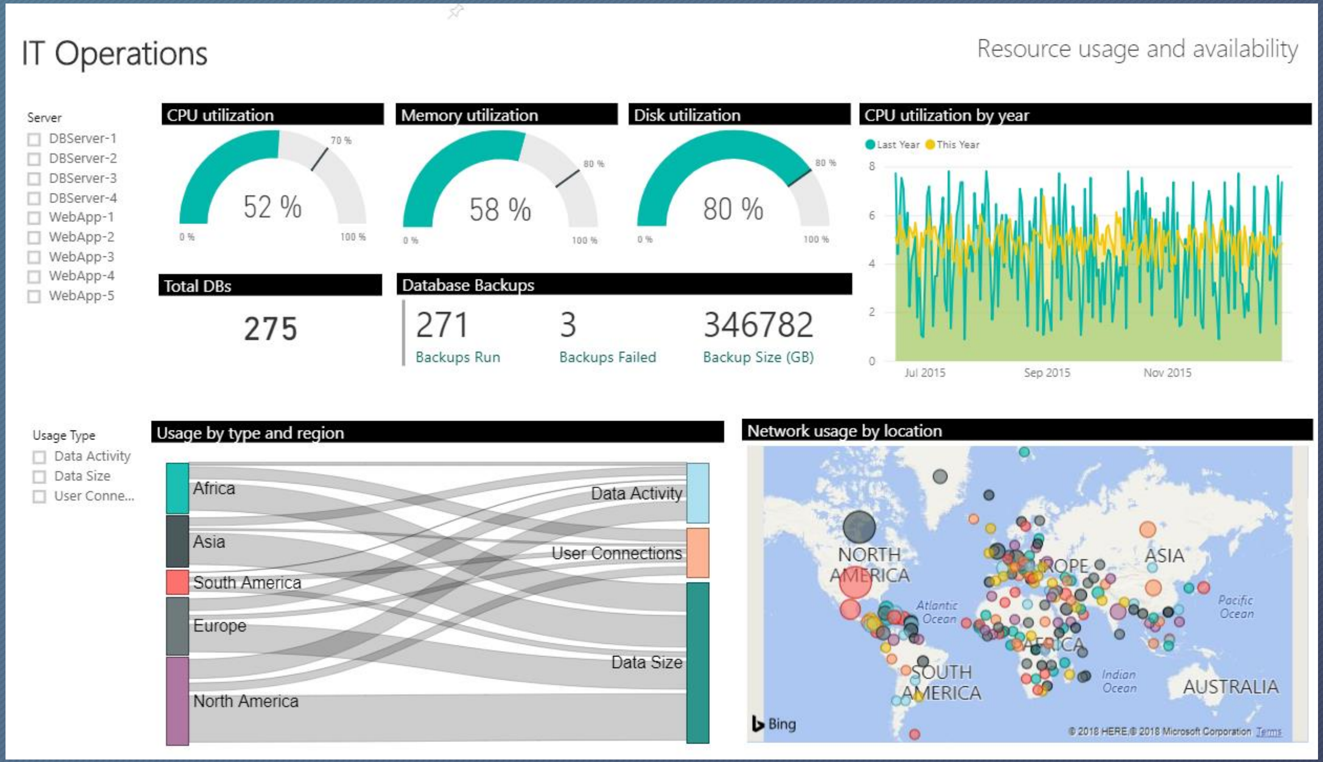
Azure Dashboard

- Visualizations such as charts and graphs can help you analyze your monitoring data to drill down on issues and identify patterns.
- Depending on the tool you use, you may also share visualizations with other users inside and outside of your organization.
- Azure dashboards are the primary dashboarding technology for Azure.
- They're handy in providing a single pane of glass over your Azure infrastructure and services allowing you to identify critical issues quickly.

Azure Dashboard



Power BI


IT Operations
Resource usage and availability
Server
DBServer-1
DBServer-2
DBServer-3
DBServer-4
WebApp-1
WebApp-2
WebApp-3
WebApp-4
WebApp-5
CPU utilization
52 %
Memory utilization
58 %
Disk utilization
80 %
CPU utilization by year
Last Year
This Year
Jul 2015
Sep 2015
Nov 2015
Total DBs
275
Database Backups
271
Backups Run
3
Backups Failed
346782
Backup Size (GB)
Usage Type
Data Activity
Data Size
User Conne...
Usage by type and region
Africa
Asia
South America
Europe
North America
Data Activity
User Connections
Data Size
Network usage by location
NORTH AMERICA
SOUTH AMERICA
EUROPE
AFRICA
ASIA
AUSTRALIA
Atlantic Ocean
Pacific Ocean
Indian Ocean
Bing