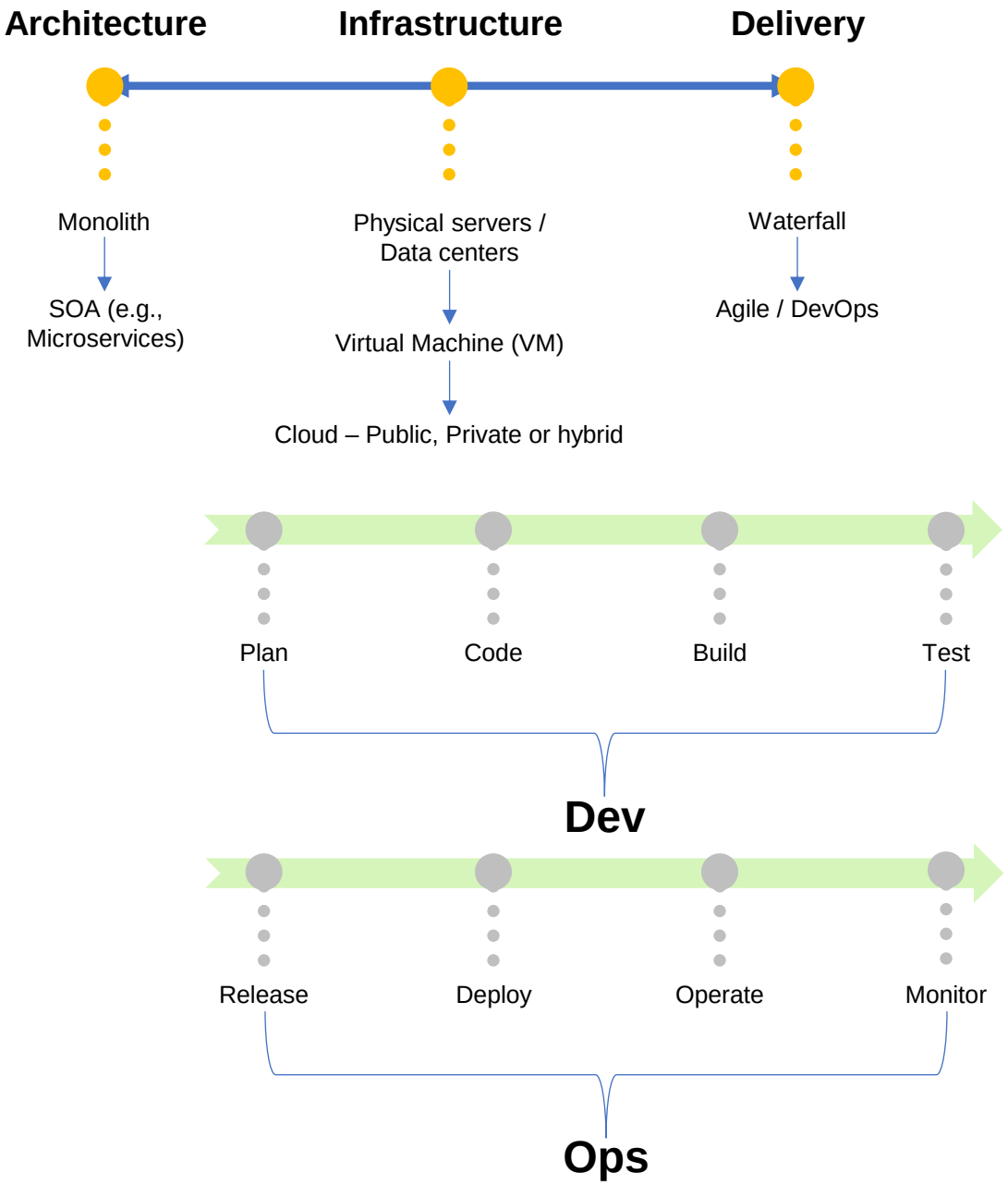
Application Development, Digital Modernization & Transformation

IT Modernization

IT Modernization – Fundamentals in an Agile World



DevOps	Agile
Integrated work culture	Software development philosophy
Automation	Sprints
Deployments	Delivery (Lean)
Dev meet Ops	Microservices
Fast flow	Cloud
Lean	Team of 10 or less
Microservices	2-4 week sprints
Cloud	Collaboration
Everyone is involved	PI planning (12 weeks or 1Q)
Many releases (100s of 1,000s)	Retrospectives
All-in-one – e.g., requirements, design, dev, testing, UAT, release, deployment, operations, monitoring.	Sprint: e.g., req, design, dev, testing, UAT, release
Continuous integration / continuous deployment (CI / CD)	Continuous Delivery

Tools & Technologies

Core repositories / Version control: Git, GitHub.

Containerization: Docker, Kubernetes.

Continuous development / integration: Jenkins.

Configuration management: Chef, Puppet.

Public cloud: AWS, Microsoft Azure, Google Cloud Provider

IT Modernization – Technology Architecture

1-Enterprise Architecture

-Agile / DevOps
-GAO | TOGAF | DODAF | FEAF | Zachman

8-Security Architecture

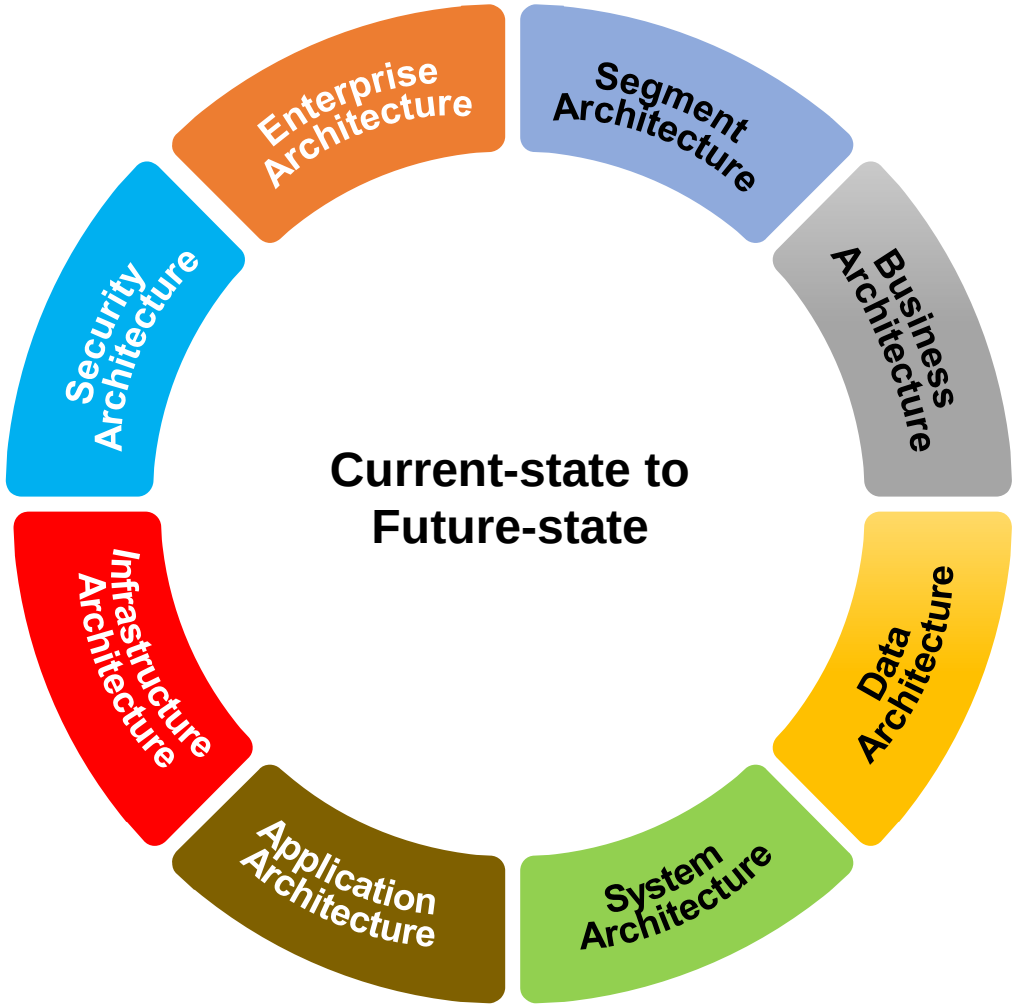
-CISO | Cyber | Cloud | Zero-Trust
-Multifactor authentication | Databases
-ZT 1-Insider threats 2-Hybrid workforce 3-Privary 4-Cloud

7-Infrastructure Arhictecture

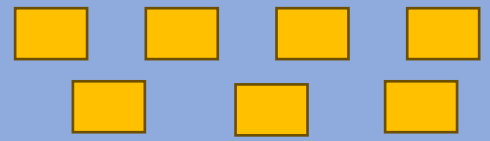
-Cloud | Cyber | Engineering
-Data centers: On-premise; Multi-cloud (public, private)
-Digital | Platform
-Hardware <> Networks <> Software

6-Application Architecture

-Sofware architecture | Design of application
-Monolithic | Layered | Microservices
-Programming languages: Kubernetes, Python, Java, .NET
-Code the application - APIs



2-Segment Architecture



3-Business Arhictecture

-Business objects
-Blueprint | Requirements
-Bridge between enterprise and system
-Business analysis
-Domain SME

4-Data Architecture

-Data modeling (micro) | Architecture (macro)
-Dataflow | Data diagrams
-Data migration
-Disparate data

5-System Architecture

-Solution architecture | System objectives / IT architecture; Solution the business problem (structure; reuse, alignment)
-Tech stack | Databases | Architecture | Standards | Domain SME





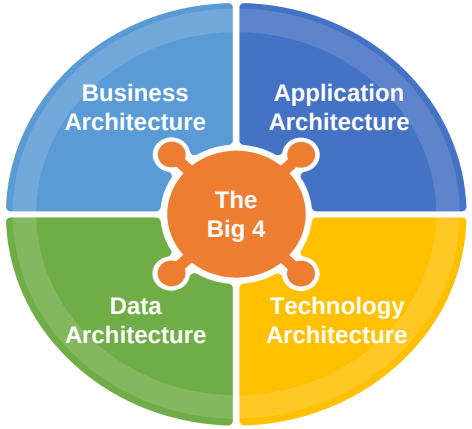




1 **Business >>> Technology**
2 **Data >>> App**
3 **Application Tech**
4 **Cloud / Tech**

Digital Modernization – Enterprise Architecture

Business processes, organization, people, strategy, information, technology



Platform services, structural maps of software application design & interactions

Data, information, databases, logical / physical data flow

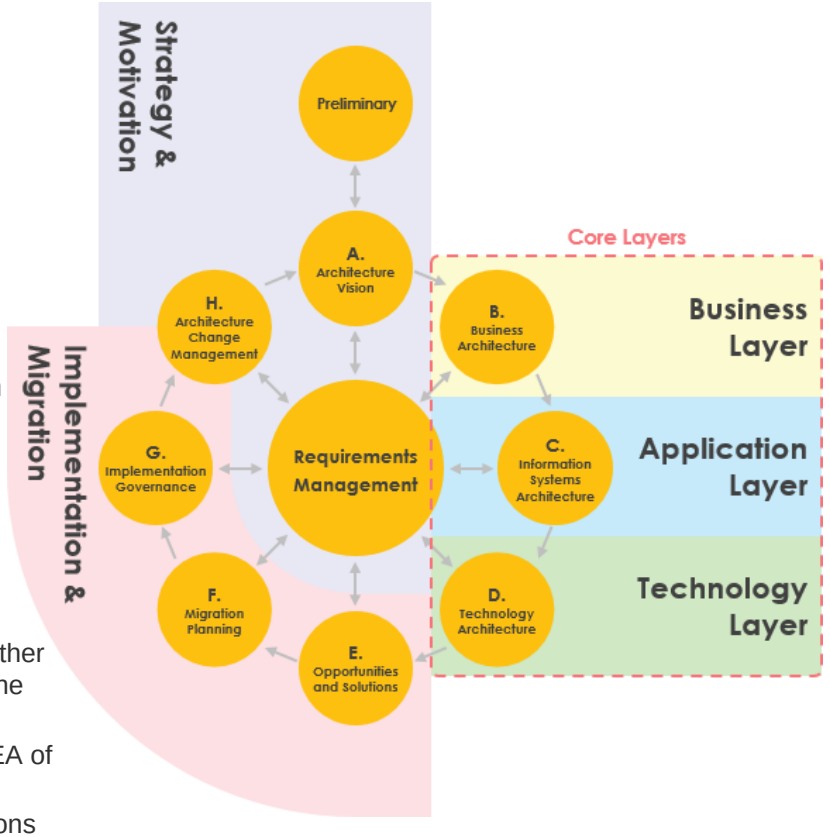
Hardware, software, networks, cloud, systems etc that support the business enterprise

Enterprise Architecture

- Align enterprise strategic initiatives by developing a defined business architecture.
- Business architecture, in turn, is used to develop technology architecture which formalizes enterprise vision.
- Critical to business survival & success; faster time to market.
- Enables innovation within the enterprise through digital transformation; efficient IT operations.
- Laws / regulations mandate use of EA – e.g., Sarbanes-Oxley, Clinger-Cohen Act (IT mgmt reform).

Architecture Development Methodology (ADM) Framework within TOGAF

- Iterative guidelines to shape the enterprise architecture
- **Preliminary** – Environment, management commitment, scope, principles, governance structure.
- **One** – Set an **architecture vision**. Scope, constraints, expectations.
- **Two** – Develop **business architecture**. Organizational structure, goals & objectives, structure, processes, functions / roles & responsibilities.
- **Three** – Create **info systems architecture**. Data and application architecture. AKA systems architecture.
- **Four** – Develop **technology architecture**. Hardware, software, cloud, communications, principles of design & evolution.
- **Five** – Create **opportunities & solutions**. Implementation planning (e.g., incremental, big-bang etc), architecture roadmap, approach (e.g., make, buy, re-use, COTS, open source).
- **Six** – Execute **migration planning**. Cost / benefit analysis, risk assessment, implementation / migration plan.
- **Seven** – Add **implementation governance**. Architectural oversight of implementation, governance.
- **Seven** – Infuse **architectural change mgmt**. Monitoring EA and change mgmt. Monitor business, capacity, communication, change.

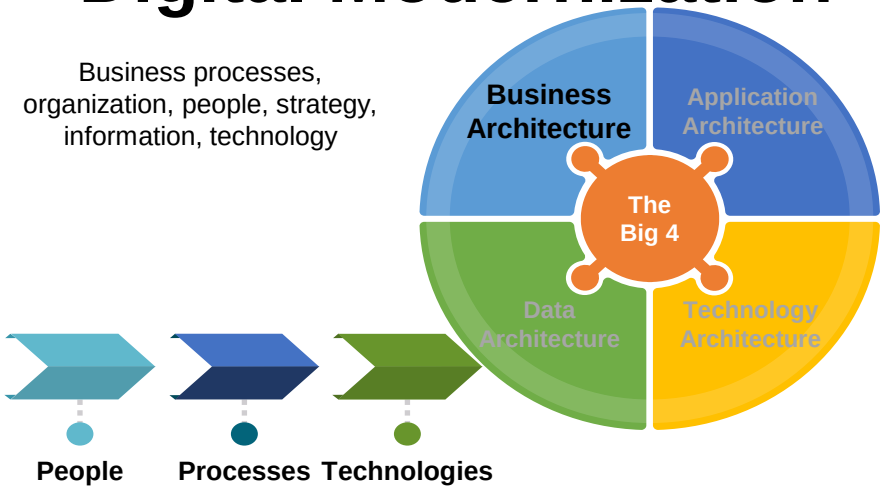


Example: Digital Twins EA

- Digital Twins (DT) are virtual models of physical systems or objects that use machine learning algorithms to process sensor data and identify patterns. Example: Tesla is a physical object, a vehicle. The sensors in the Tesla represent the digital view of your car. The sensors can help gather data to improve driving conditions, battery life, crime reduction, car insurance statistics etc. Digital Twins are the new enterprise architecture of the physical model.
- DTs can be developed for the organization's EA. EA is static and used for current-state to future-state solutioning. DTs can shape the future EA of the organization – e.g., track supply chain, use data to build future-state models.
- Healthcare – Use DTs to enhance patient care, optimize provider platforms, train staff, reduce unnecessary procedures, plan hospital operations and supply chain, replicate medical devices etc.
- Modeling examples include physical, abstract and digital types.

Digital Modernization – Business Architecture

Business processes, organization, people, strategy, information, technology



Business Architecture

- Business architecture is the blueprint of your organization and responds to executive, stakeholder, employee and customer needs.

Customer request for product or service

Operating model	Access channels; processes; people; applications; data; product.

Executive viewpoint of customer needs

Capability model	Enable operating model by focusing in on enabling capabilities. Which capabilities will effectively serve your clients.

Operating model *when you have burning customer needs;* ***Capability model*** *when you are focused on emerging needs.*

Business Architecture

- Enterprise architecture is enabled by a strong business and technology architecture. **Business architecture** defines how a business operates (e.g., projects, products, roles / responsibilities, processes etc). **Application architecture** defines feature-driven, Human-centred Design (HCD) systems that address business needs (e.g., modernized software). **Data architecture** is a gold mine for digital transformation (e.g., petabytes of data). **Technology architecture** rounds out the big 4 with defined hardware, software and networks. As an example, in the new world of DevOps CI / CD, cloud is king. Cloud vendors want to take all the complexity away from you so that you can focus on your business. But organizations must develop this EA to fully understand and be able to manage their business. The cloud vendor manages your architecture including your data and security.

Capabilities Model

- What competitive differentiators does the organization have to offer – e.g., fastest delivery, supply chain, advanced digital capabilities, better products / services.
- A set of capabilities pulled together helps deliver value to customers. Capabilities can have small or large components. Capability requirements will change based on emerging market needs – e.g., competitive landscape. Capabilities must align with strategic vision / goal of the organization. Capabilities must address the strategic vision. Note: Capability modelling can be done using process mapping (current-state gaps to future-state solutions). Current-state capabilities map leads to a future-state roadmap.
- Example of Agile capability mapping: Epics (L1) -> Features (L2) -> User Stories (L3) -> Use Cases (L4).

Value Stream Mapping (VSM)

- VSM shows an end-to-end set of values (e.g., value chain) without any wasteful steps. VSMs can be used in business architecture, Agile application development (e.g., DevOps), customer product / service alignment, manufacturing (e.g., Toyota Production System) etc.
- VSMs are a great way to create a “one vision” future-state viewpoint of the organization (i.e., current-state pain points → future-state solutions). VSM is executed across segments / BUs. It is a “happy path” solution.

Design, Systems & Architectural Thinking

- **Design Thinking** – In a hospital setting, look at the full perspective (e.g., staff expertise, right diagnosis, right medication, emotional comfort, ambiance etc). Understand context, prototype solution. Use Human-centred Design (HCD) to create design.
- **Systems Thinking** – Focuses on connections, interactions and dependencies between different parts of the system. How will the hospital systems be developed to ensure design thinking is built into the applications?
- **Architectural Thinking** – Takes a broader view than systems thinking, considering the overall structure and principles guiding software development.

Role of the Business Architect

- Understand the customer’s current-state problem. Gather all relevant documents – e.g., data stores, documents, human knowledge, presentations etc. Frame a mental model and understand the business strategy and technology that will drive change.
- Business architect must be a master “story-teller” (e.g., process mapping) so that everyone else can understand the complexities of the current-state and how to develop a future-state solution.
- Digital transformation – Business architects serve as digital transformation advocates. Be an internal champion working closely with technology SMEs, consultants etc to develop the best digital transformation strategy.

Business architecture: *Identify, understand, visualize and optimize the blueprint…business architects contribute expertise in strategic alignment, process design and organizational structure…*

Digital Modernization – Business Architecture

Innovation through Business Architecture

- Business needs creativity and innovation, as well as robust architecture.
- Product innovation – Create products and services that customers love and scale them exponentially. Creativity leads to new, unique ideas.
    - Design, system and architecture thinking leads to product innovation.
- Cultural innovation – Tesla is a great example of cultural innovation. Customers buy Teslas because it makes them happy.
- Architecture innovation – is about the design and construction of products and services.
    - Architecture enforces modularity, standardization and integration to compose innovative products.
- Example: Atlassian let's their employees spend 20% time on innovation.
- Architecture is a combination of "right brain" innovation and "left brain" structure. Innovation and creativity leads to optimal architecture.
    - Creativity + structure = success!

Business Architecture within Product Management

- Product management requires an Agile mindset of continuously developing and deploying products for an organization. Enterprise and business architecture enables the optimal product management strategy.
- SAFe loosely addresses business architecture within portfolio management activities, helping translate strategic themes into Epics / user stories which provide business context.
- Product managers / owners focus on products while business architects should be looking at a broader, enterprise view. Best practices would be to map business strategies, capabilities and value streams to product development.
- Business architects can play a key role in product management. Product managers set the vision and maximize the value of products while product owners oversee the development and evolution of products. BAs build the blueprints for products (e.g., capabilities, roadmaps, technologies etc).
- Business architects can also serve as SMEs for scrum masters during product implementation to ensure delivery of business objectives through technology.
- Business architects are proxies between enterprise strategy and product. The BA my not have full direct access to the customer but the product manager will have that viewpoint.

Business Process Diagramming

- Types: Process flow diagram / process map (e.g., 3 or 9 steps), flow chart (e.g., vertical), capability map, value stream / value chain (e.g., where is the value provided), cross-functional diagram (e.g., swim lanes), business context diagram, activity diagram, customer experience map (e.g., CX, HCD), state diagram (e.g., current to future-state), systems map, doodle (e.g., create your own design).
- Process modelling – Gauge level of detail required.

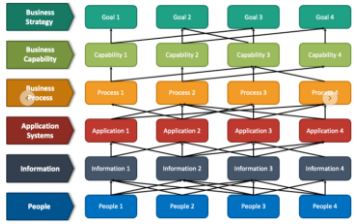
Process Flow Diagram



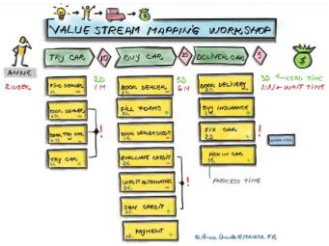
Flow Chart / Process Map



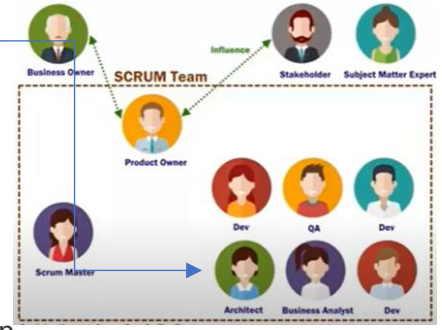
Capability Mapping



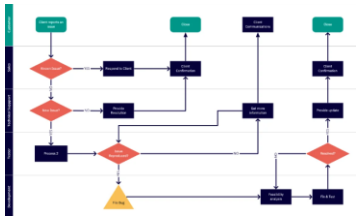
Value Stream Mapping



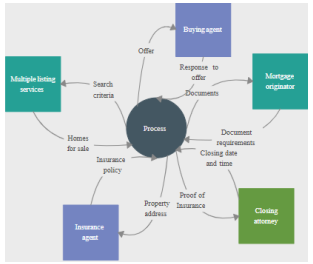
Scrum Team



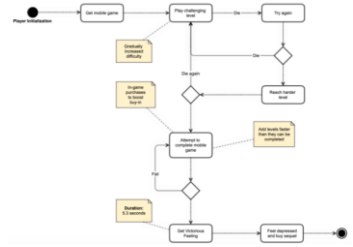
Cross-functional Diagram



Business Context Diagram



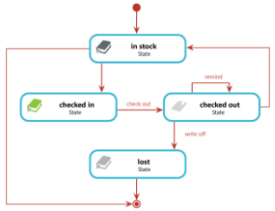
Activity Diagram



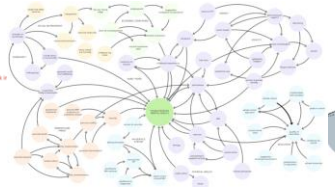
Customer Journey Mapping (CX)



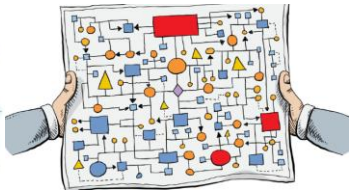
State Diagram



Systems Map



Doodle Your Own



SAFe – Scaled Agile Framework (IT)

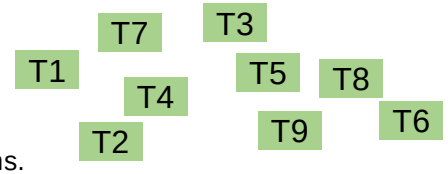
Lean/ITIL (80s) > Agile (2001) > DevSecOps (2008)



Agile Teams

-5-11 FTEs
-Iterations (2-4 week sprints)
-Backlog – Epics, features, user stories, use cases

Multiple Agile Teams



-Agile Release Trains (ARTs)
-50-125 FTEs total across teams.
-SAFe roles: Release Train Engineer (RTE); Scrum Master for each team, Product Owners

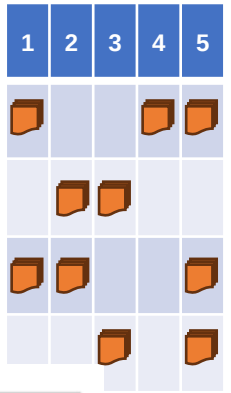
Scrum-of-Scrums

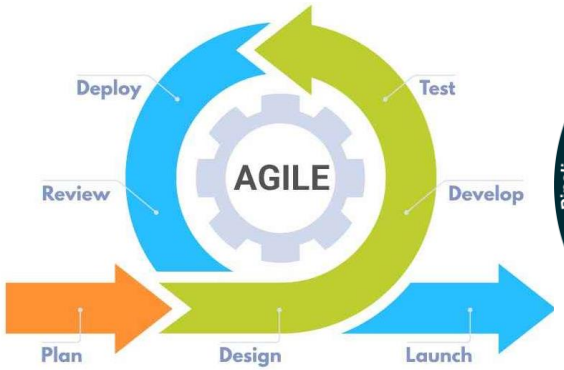
T1	T2	T3	T4	T5	T6	T7	T8	T9
SM	SM	SM	SM	SM	SM	SM	SM	SM

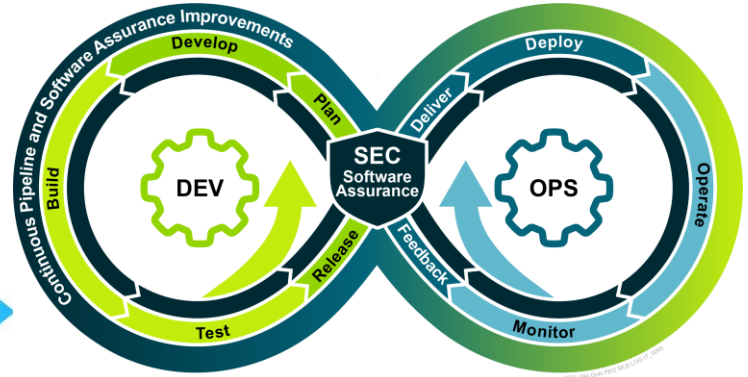
PI Planning for all Teams (takes 2 days to plan)

Q1	Q2	Q3	Q4
8-12 Weeks	8-12 Weeks	8-12 Weeks	8-12 Weeks
4-6 Iterations	4-6 Iterations	4-6 Iterations	4-6 Iterations

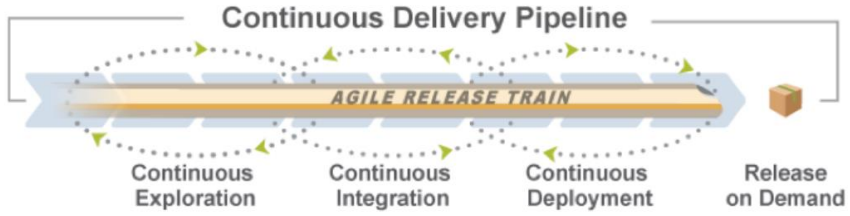
5 Iterations







Agile Release Train (ART)



-Retrospectives after each Sprint
-Execute iterations Q1-Q4 (Go-live) - Sprints
-Continuous delivery (e.g., CI / CD)

DevSecOps

Cloud Computing

Cloud Deployment Models

Public

-Like a bus

-Service providers - AWS, Google Cloud Provider, Microsoft Azure

Private

-Like a car

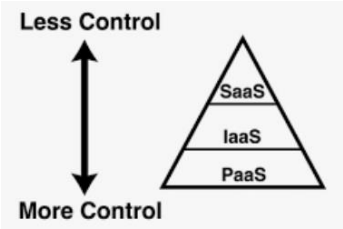
-One company could be hosted by a 3rd party

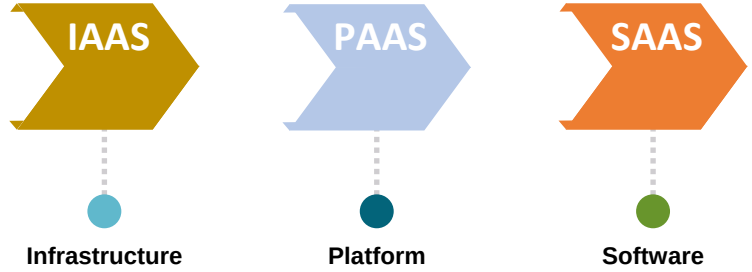
Hybrid

-Both public and private

Multi-Cloud

-An organization may use cloud computing services from at least two cloud providers to run their applications



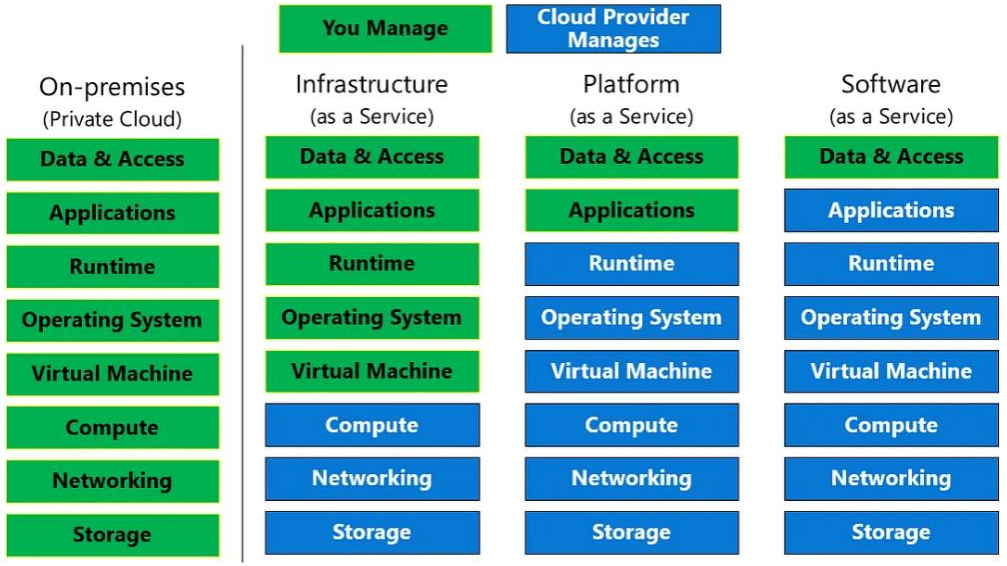


On-premise

-No cloud, data centers, hardware = you manage everything!

License / Rent	Scalability	Globalization	Agility	Cost Savings (long-run)	Physical Space Saving
Pay as you need only	Data storage (e.g., petabytes)	Worldwide – e.g., US, UK, Japan, Brazil	Ramp-up very quickly; stay nimble	Costs may be higher than on-prem to start but should go down over time.	Limited hardware

Front-end	*Back-end*
-Software -Client devices (e.g., phone, laptop etc) -User interface; GUI -User interface / experience (UI / UX)	-Computers, servers, data storage (remote) -Cycle: Applications, services, cloud runtime, storage, infrastructure, middleware, security



Cloud Computing

Virtualization

- Enables the cloud
- Compute, storage, networks, applications, servers

Physical server (host O/S) → Hypervisor (virtual) – software that allows you to run multiple virtual machines on a single physical machine. → **Cloud**

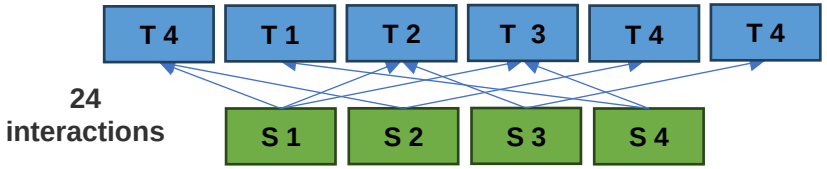
Multiple Hosts

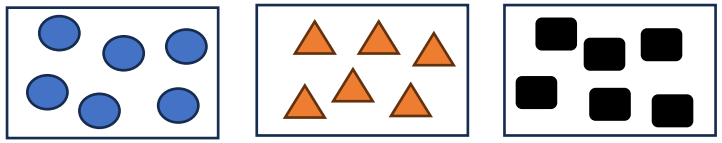


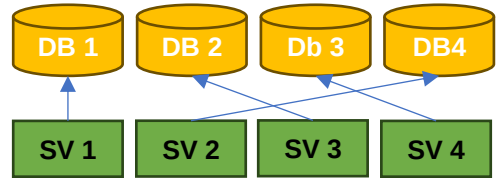
Apache Kafka

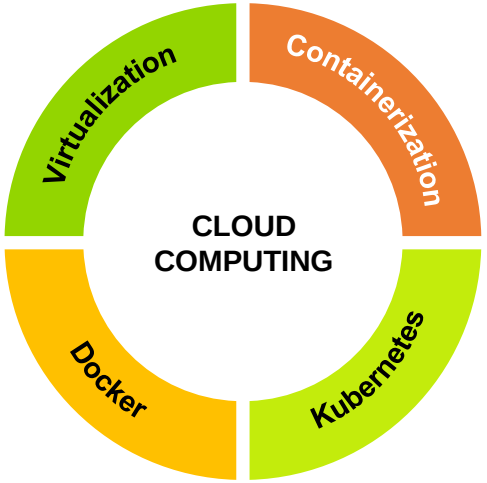
- High-throughput messaging system; Message broker system

(1-Distributed 2-Resilient Architecture 3-Fault Tolerant)



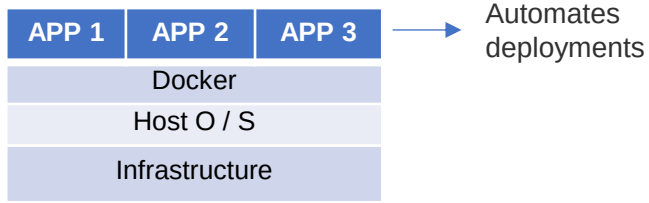






Containerization / Docker

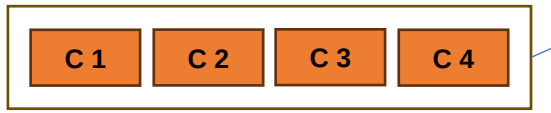
- Docker enables containers (i.e., containerization framework)
- Purpose: Pack / ship apps
- C1 C2 C3 (containers); runs on single nodes
- Lightweight, isolated, cloud-agnostic, virtual, microservice…
- Decentralized; 1,000s of microservices
- Python, Java, .NET, JavaScript, Ruby



Automates deployments

Containerization / Kubernetes

- Microservices enables decentralization; groups containers
- Purpose: Deploy & scale apps
- Kubernetes is used to deploy containers
- Python, Java, .NET, JavaScript, Ruby
- Runs across clusters



PODs (Microservices)

DevOps / Continuous Delivery

DevSecOps (Handbook)

Lean/ITIL (80s) > Agile (2001) > (DevSecOps)

Best Organizations	
1) Etsy	4) Nordstrom
2) LinkedIn	5) Target
3) Netflix	6) Capital One

-Automation – Scale developer productivity – Automated Testing – CI / CD

-CI / CD - Infrastructure-as-Code (IaC): Pre-configured code / Environment (for VM, network, topology etc.

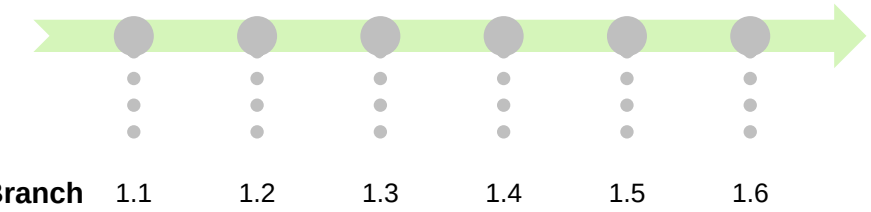
-Source code to production

-Value stream mapping – Fail early / Fail often

DevOps Big 4

1) **Deployment frequency**
2) **Lead time for changes**
3) **Change failure rates**
4) **Mean Time to Resolution (MTTR)**

Trunk / Master



Local Development to Trunk

Developer 1 – Building in 1.2.1 via branch
Developer 2 – Building in 1.2.2 via branch
Developer 3 – Building in 1.2.3 via branch

Merge all to Trunk as 1.3

CI/CD – 100s, 1,000s, 10,000s of deployments

DevOps for Bimodal IT

1-System of Record (ERP, EHR). Human Resources (HR); Financial Services (Oracle, SAP)

-Records, few changes to system, compliance (e.g., SoX)

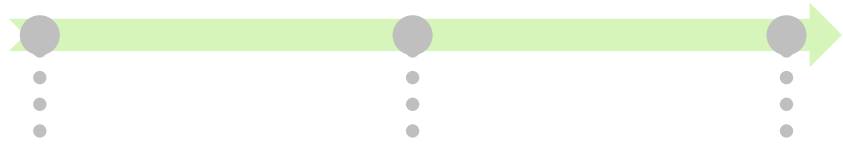
"Do it Right" ✓✓✓

2-System of Engagement (CRM). eCommerce

-Client-facing, customer service

"Do it Fast" ✓✓✓✓✓✓

Lead Time



Feature creation >>> Design

Processing time >>> Dev

End-user readiness >>> Prod

Tools & Technologies

Tools for DevOps: Need unified tool (one backlog)

1-Development/PM: JIRA **2-Operations**: ServiceNow

Chat Rooms: Slack, Hipchat, Openfire

Technical Debt reduction (allow developers to work on below)

1-20% DevOps efforts on non-functional requirements + refactoring + automation (reduce tech debt ↓)

2-Stop development to fix tech debt >>> core infrastructure (e.g., LinkedIn Project Inversion)

Conway's Law – Designs match the make-up of the organization; Large org = less flexible design with tight coupling and small changes can have major effects.

DevSecOps (Handbook)

DevSecOps – A true value stream

-Product Owners - Developers - DevOps Engineers - UI/UX - QA/QC - Operations - InfoSec Engineers - Release Managers - Value Stream Managers - Site Reliability Engineers (SREs)

Types of DevOps Execution Environments

1) Functional (by Function) – Optimize cost
2) Matrix (by function + market) – Customer-focused
3) Market (by market) – Quickest to customers
-Flat (marketing, finance, HR, IT, engineering, accounting etc. Think Amazon or Netflix.
-Optimize speed; full DevSecOps.

Development	Operations	Security	Testing
-The “doers” – QB, WR, RB -Loosely coupled architecture -Short lead times -100s, 1,000s of developers -Services – SOA -Service #S1, S2, S3, S4 -Independent + loosely coupled -Service Oriented Architecture (SOA) – Flexible / scalable -Think Google or Amazon -2 pizza teams -Small with clear / shared goals -Decentralized power -Autonomy -APIs - Single source of truth -Organization	-The “guardians” – Offense linemen (project) -Dev meets Ops (Shift left) -Key roles: Business relationship manager; Release Train Engineer (RTE) -Cycle: Developers >>> Release Engineer >>> SREs -Self-service (by market) 1-increase developer productivity 2-Embed operations engineers 3-Assign operations liaisons -Shared version control -CI / CD -Same tools / technologies (Jira, ServiceNow) -Embed operations engineers into service / product teams -Operations teams (product function, operations, scalability, observation, monitoring. -Ops engineers join standups, ceremonies, retrospectives (e.g., Kanban)	-**Traditional** 1-Segregation of duties 2-Change approval 3-Security; -**DevSecOps** 1-Integrated controls 2-Integrated security -Security + compliance throughout SDLC -Put security into defect tracking **/** post-mortems -Jira - Review all security issues -Add preventive security shared source code 1-Authentication 2-Encryption 3-Configuration hardening 4-Database security 5-Libraries -Deployment pipeline automation -Security testing at every code commit -Happy path / sad path -Ensure security of applications / platforms / environment (e.g., monitoring, logging, observability, alerting -Add security into production telemetry -Security telemetry (Dev, QA, Ops) A-Security telemetry for applications 1-Log-ins 2-Passwords 3-Email 4-Credit cards B-Security telemetry for environments 1-Configurations 2-Cloud infrastructure 3-Web server 4-SQL Injection	-Automation and integration -Testing types 1-Unit 2-Integration 3-Load 4-Smoke 5-Regression 6-Stress 7-Acceptance 8-Capacity 9-Performance / Stability 10-Exploratory 11-Functional 12-User Acceptance Testing (UAT) using Personas. Test-driven development 1-Create test scripts 2-Write code Code Review 1-Pair programming 2-Over-the-shoulder 3-Email pass around 4-Tool assisted -“Shift-left testing” – A/B testing -Automate testing 1-Unit 2-Acceptance 3-Integration -Manual testing 1-UAT 2-Exploratory 3-

Tools & Technologies

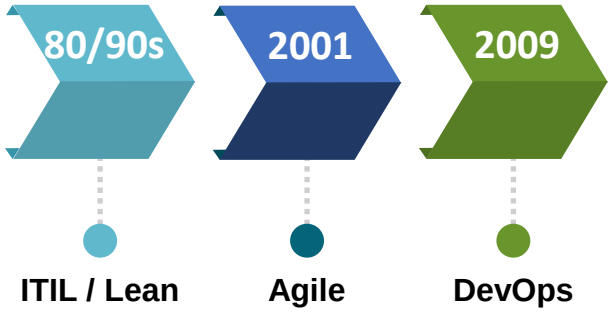
Tools for Development / Operations: Jira (Atlassian). Confluence, Jazz, ServiceNow

Tools for Security: Prometheus, Grafana, SolarWinds

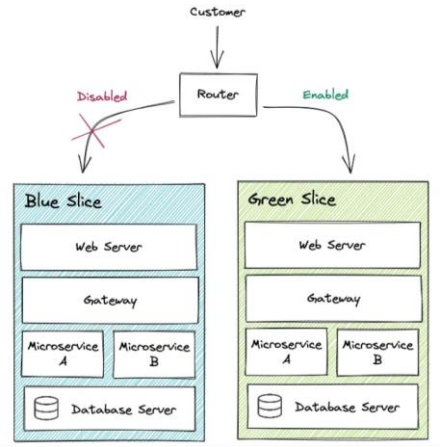
Tools for Testing: Selenium (Apache license 2.0)

DevSecOps - Concepts

DevSecOps – A true value stream



Best DevOps Orgs	
-Netflix -Google	-Etsy -Amazon



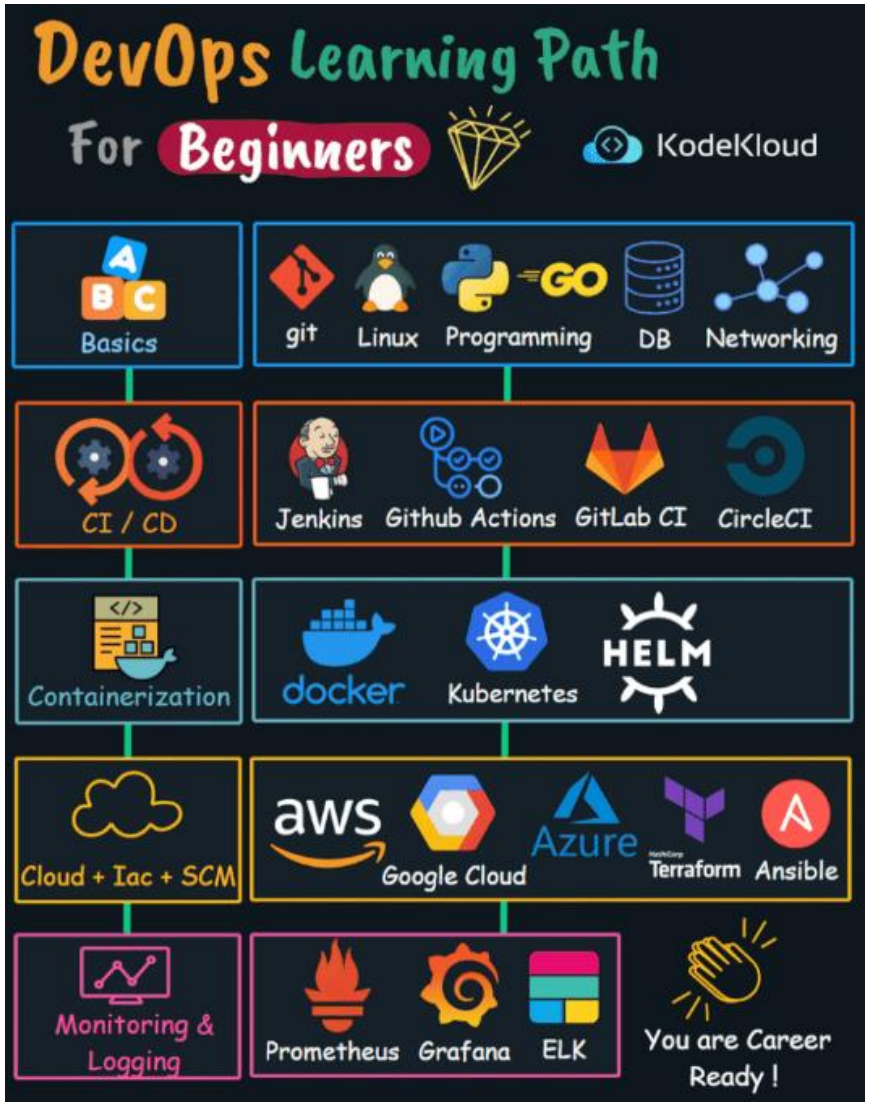
Product / Feature-driven Development (best practices)

- Automation (CI / CD) – free technical debt
- Loose architecture – Metrics – MTTR
- Developer productivity – short lead times
- Fast flow – continuous feedback, continuous learning / experimentation
- Small batch sizes – single piece flow (F1 L1 A1 C1 **not** F1 F1 F1 L1 L1 L1)
- Consistent, matching environments – self-service deployments
- Deployments (i.e., code / version deployment to PROD) ≠ Releases (i.e., new feature-sets for customers)

Environment-based Releases
-Canary – Progressive roll-out of an application to a sub-set of users (i.e., put the feelers out there). -Blue / Green – Deploy to 2 similar environments, which sending traffic to only one. Blue will run the current application version, green will run the new application version. -Microservices - By unit, scale, performance, test / deploy. -Production telemetry. -No change to application code.

Application-based Releases
-Feature toggle: 1-Roll-back changes as needed. 2-Degrade performance if needed (e.g., reduce the number of customers who can access a certain feature, disable CPU-intensive features). 3-Increase resilience via SOA. Toggle features on/off based on available services. -Change to application code and then release the features to the customer.

DevOps Lifecycle – Where development meets operations



LAMP Stack	
-Linux O/S -Apache Web Server	-MySQL Database -PHP Language

Continuous Delivery - Concepts

DevSecOps – A true value stream

Patterns of success

-Pattern #1 – Deploy automatically

-A-Fewer bugs B-Repeatable process C-Deployment scripts readiness - documentation D-Free-up resources E-Audit trail

Pattern #2 – Continuous deployment to production

-Integrate testing, deployment, release into production builds

Pattern #3 – Automated configuration of production builds

-Configuration via automation A-From version control – e.g., O/S, patch levels, application stack, infrastructure configuration B-Changes made in version control and propagated to production through automation

Automate everything

-Code verification -Application configuration -Security -Code merge to trunk -Database

-Testing 1-Integration 2-Regression 3-Load 4-Performance 5-UAT

And…automate everything with version control

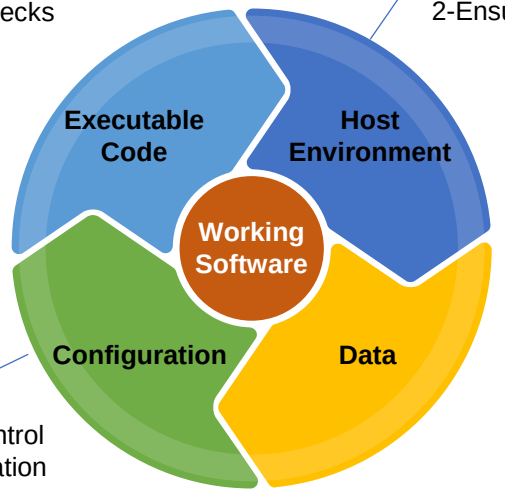
-Virtualization or hardware >>> Puppet, Chef

-Network / firewall configuration

-Software release / deployment

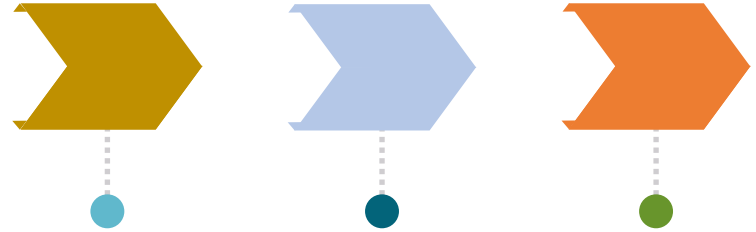
-Post deployment checks

1-Environments should not be different >>> mirror each other
2-Ensure portability



1-Practice version control
2-Automate Configuration Management (CM)

Deployment Pipeline

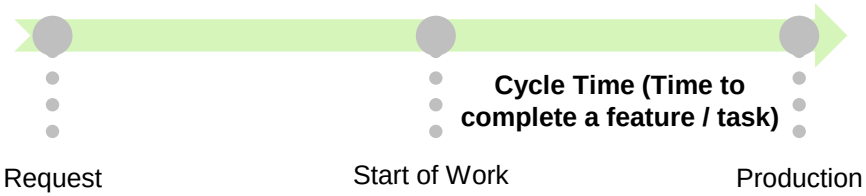


Collaboration – Build, Deploy, Test, Release

Fast Feedback Early & Often

Any Version/ Any Environment

Lead Time (Time to deliver a requested feature)



Frequent + Automated + Fast Feedback and…Version Control (Build, Deploy, Test, Release)

Always remember…

-Software releases should be low-risk, frequent, cheap and predictable

-For small start-ups to world-wide enterprises

-Push button software releases

Deploying software

1-Provision / manage your environment (hardware/software, infrastructure, external servers)

2-Install correct version of your application

3-Configure your application – e.g., data

Release candidates

-Every code base change is a release candidate

-Software is always working (working software)

Tools & Technologies

Tools for Virtualization/CM: Puppet, Chef

Continuous Delivery – Configuration Management

Configuration Management = Version Control

-New features >>> Commit / check-in often – Continuous code merge into trunk

-Binary files (8 bits each) >>> 1-Executables 2-Libraries 3-Graphics 4-Databases 5-Archives

-CI / CD >>> 1-Reduce cycle times 2-Increase quality 3-Automated testing 4-Push button deployment

Configuration Management Up-to-date via Version Control
1-For Analysts >>> Requirements docs, functional specifications
2-Testers >>> Test scripts
3-Project Managers >>> Release plans, risk log

Types of Configuration Items (CIs)	Application Configuration Storage	Add New Configuration Items (CIs)
A-Your build scripts can pull configuration in and incorporate it into your binaries at build time. B-Your packaging software can inject configuration at packaging time, such as when creating assemblies, ears or gems. C-Your deployment scripts can fetch the necessary information and pass it to your application at deployment time as part of the installation process. D-Your application itself can fetch configuration at startup time or run time. #C / D are the best time for binaries. Use same binaries in every environment.	A-Store in 1-Database B-Version control C-Directory or registry D-Should be same repository as source code to match CM to application B-Tools: YAML in Ruby, JAVA, XML, Restful Web Server	1-Adding a new environment – e.g., for capacity testing. 2-Creating a new version of your application 3-Promoting new version of your application from one environment to another 4-Relocating your database server - i.e., reporting database via CM 5-Managing environments using CM virtualization – i.e., VM configuration. Virtualization = New environment ready at the click of a button

Make your production configuration the default

-Use a firewall to ensure production doesn't get hit by mistake.

-Any environment specific tailoring is reduced to only those configuration properties that must be changed for the software to work in that particular environment.

Testing system configuration

-You should be able to ping all external services (e.g., Messaging bus should be up and running during deployment).

-Run smoke tests once application is installed (e.g., Run few tests to ensure functionality dependencies are working with CM items.

Configuration Management (CM) visibility – Managing configurations across applications

-CM visibility via production monitoring system (i.e., operations).

-The goal is to be able to see each application's configuration through your operation team's production monitoring system, which should also display which version of each application is deployed in each environment.

-Tools such as Nagios, OpenNMS and HP OpenView all provide services to record such information.

-Reproduce your environments through repeatable CM automation.

-Environment configuration items 1-O / S 2-Supporting software packages 3-Network topology (separate CIs and binaries 4-External dependencies 5-Production data.

Tools & Technologies
Tools for configuration storage: YAML in Ruby, JAVA, XML, Restful Web Server
Tools for application configuration monitoring: Nagios, OpenNMS and HP OpenView
Tools for O/S configuration automation: Puppet, CF Engine

Continuous Integration (CI) – CM to CI

Version Control → An Automated Build → Agreement of Team

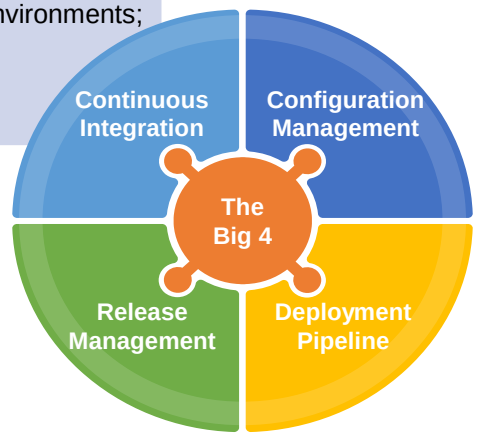
CI Items

-Source code, build scripts, requirements, data, libraries etc.

-Development, testing and operations tools; All environments; Application tech stack

-Configuration – All applications + environments

-Build, deploy, test, operate



Continuous Integration Best Practices

A-Check-in frequently to trunk – don't' even use branches

B-Create full automated test suites – unit (specific function), component (several components), acceptance (full functionality, test various functions, non-functional items)

C-Keep build and test process short

D-Manage development workspace >>> Environments (1-local 2-CI 3-Testing 4-Production

-Configuration management of source code, test data, database scripts, deployment scripts + 3rd party libraries and dependencies / components

-Smoke test for readiness; Test automation; Version control; Known good version; Ready to commit and check-in

CI server build status (important!)

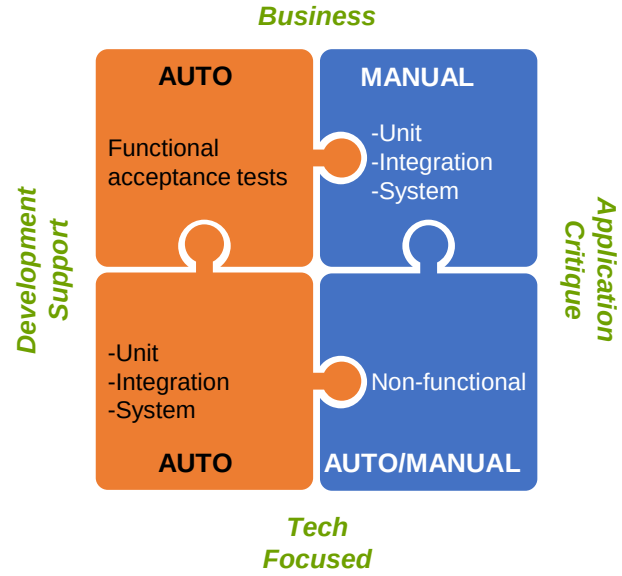
- Catch failures prior to PROD
- CI server >>> PM tracking tool (JIRA)
- Don't check in on a broken build (Always green / working software)
- Run commit tests locally (Release candidates)
- CI server for local build success

Broken build?

-Fix (timebox it) or revert (go to previous version) >>> don't push to PROD

-Don't comment out failure tests; Watch for regression bugs

-Use test-driven development >>> Prior to a new feature development, write an executable test script (1-Proves expected outcomes / hypotheses 2-Helps with regression testing)



Testing Strategy	Continuous Integration (Core DevSecOps)
-Automated testing via continuous integration leads to better regression testing, functional testing, non-functional testing. -Testing tools (e.g., Cucumber) enables automation for functions. -Automated acceptance tests 1-Fast feedback loops 2-Reduce load on testers 3-Testers can focus on exploratory / high-value tests 4-Allows for regression testing 5-Requirements become test scripts. ***Unit + Component + Acceptance Tests*** = 80% test completion (all automated)	-Continuous integration is part of Extreme Programming (cir 2000s) -CI + Test-driven development enable small incremental changes + refactoring. -Check for bad code – e.g., Style, validation, errors, syntax, QA / QC, bugs, security. -Every commit should reduce errors / warnings. -Extreme programming >>> Continuous Integration means "All the Time" – Every commit goes into version control

Tools & Technologies

CI tools: Hudson, Cruise Control

CI servers: Go, Team City, Atlassian's Bamboo, Zutubi's Pulse

Testing tools: Cucumber, JBehave, Twist, Concordian

Code quality testing tools: Simian, JDepend (Java), NDepend, CheckStyle, FindBugs

Acceptance Testing

Development tests (mostly automated)

Unit Tests	Component / Integration Tests	Deployment Tests
Don't hit the database or touch file systems or external systems.	Input / output, data, files, other systems.	Application configured / installed, tests answer to services.

User acceptance tests (automated / manual)

Exploratory Testing	Usability Testing	Beta Testing
Learning process to find bugs / make application better.	Metrics, performance, delivery, timing etc.	-Release features secretly, several versions at once. 1-Canary releases 2-A / B testing 3-Blue / Green testing.
Across these areas of UAT, Functional (e.g., Features) and non-functional (e.g., capacity, performance, security, availability)		

Protect PROD data…

Abstraction Layer

Create an abstraction layer to protect the User Interface. Acceptance testing will not hit PROD data.

Public API

Build a public API below the UI that serves the UI. User Interface will not have business logic and will not hit PROD.

Testing best practices

-Testers should be helping write requirements from the start using an automated acceptance test suite. Use "shift left" testing for better code.

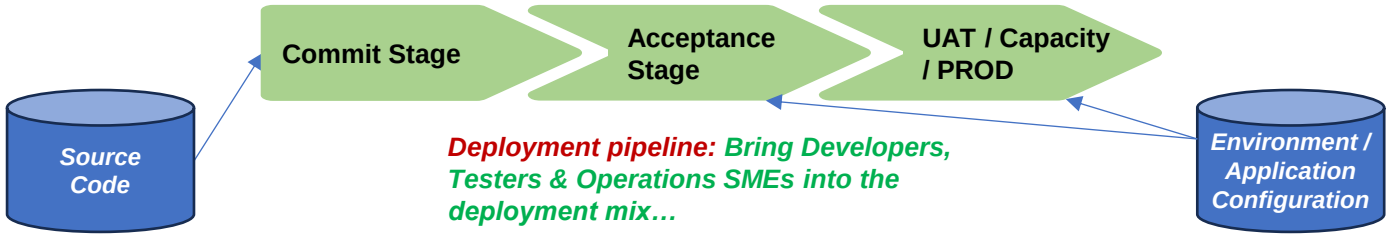
-Test simulated version of external system or firewall it! Use "test doubles" 1-Dummy objects 2-Fake objects 3-Stubs 4-Spies 5-Mocks

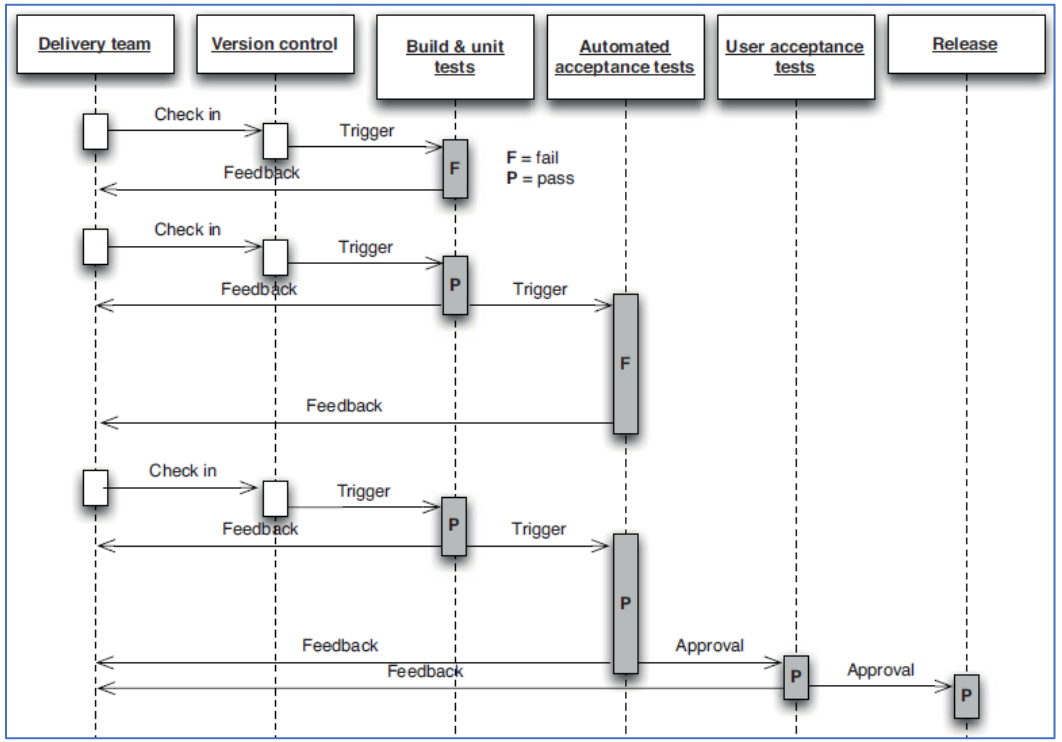
-Automated integration tests >>> Can be reused to gear up for smoke tests *and* for production monitoring.

-Test-driven development + continuous integration + automated test scripts + acceptance / unit / component tests = ZERO DEFECTS (almost)

Deployment Pipeline

Deployment Pipeline Best Practices	Value Stream Mapping (3.5 month outlook)
-Continuous Integration (CI) >>> Continuous Deployment (Testing, QA, PROD releases) with end-to-end push-button deployments.	1-Product / System assessment 2-Discovery 3-Planning / Estimation 4-Development 5-Testing / Approval 6) Product release A-Clean deployment pipeline – e.g., catch bad code, issues between components and environment, new network topology issues, PROD configuration issues. Mitigate/fix all of these issues. B-Deployment automation – e.g., reverse code anytime; deploy anytime in any environment.





Deployment Pipeline

Development tests (mostly automated) *User acceptance tests (automated / manual)*

Deployment Pipeline Best Practices					
-Build binaries (0s, 1s) only once – e.g., Jars, .NET assemblies, .so (for all environments). Executable code.	-Deploy the same way to every environment >>> one single source for your environment configuration settings – e.g., Version control, directory, database. -Test <> UAT <> PROD all the same.	-Smoke test your deployments	-Deploy into copy of PROD. 1--Infrastructure 2-O / S configuration. 3-Application stack. 4-Application data is in a known-valid state.	-Every change should propagate instantly via version control. -CI server should allow for continuous delivery (i.e., scheduling workflow).	-If any part of pipeline fails, stop the line! -Delivery - version control - build / unit tests - automated acceptance tests

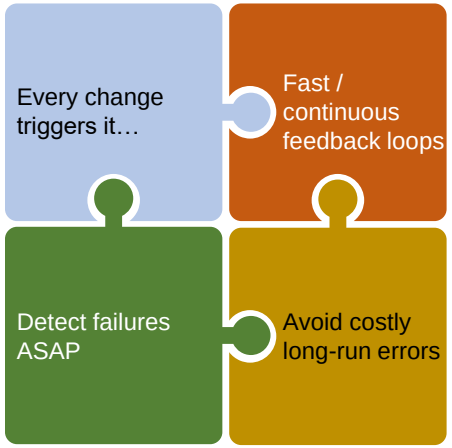
If your deployment pipeline is executed properly, you can only have face 3 issues…

1-Environment-specific application issue 2-Infrastructure issue - application deficiency 3-Configuraiton of environment.

Commit Stage

Common tasks don't change	Tasks that change often (i.e., new code base)
-Build specialist (e.g., structures, patterns). -Build master should **not** be a permanent role. Shuffle folks around.	-Dev & Ops folks (e.g., new libraries, configuration files, data changes).
-Outputs of commit stage (e.g., binaries / reports) should be stored in an artifact repository (Nexus, Maven). -Leave UI testing to the "acceptance test" stage. Inject dependencies (Example: Create CAR class and inject ENGINE).	
-Testing (Unit) – Separate code from database 1-but you should smoke test during commit 2-use test doubles such as stubbing (only test what was coded), mocking, fake tests (use test ENGINE from CAR) etc. -Perform a simulation of a real system.	
Stubbing – Set up manual fake test.	Mocking tools – Automatic fake test; call the right code.

Commits should only take 5-10 minutes! Split them into suites using CI server tools…



Every commit starts with a change

1-Version control 2-Trunk 3-CI server 4-Check source code 5-Commit tests 6-Create binaries 7-Check health of code 8-Database migration / test data artifacts.

Enter deployment pipeline

-Catch critical errors (A-syntax errors in code B-semantic error in application C-Configuration issue). Goal is to quickly fix developer mistakes.

-Commit stage fails if 1-compilation fails 2-tests break 3-there is an environment issue.

-Continuously review app quality and enforce quality metrics.

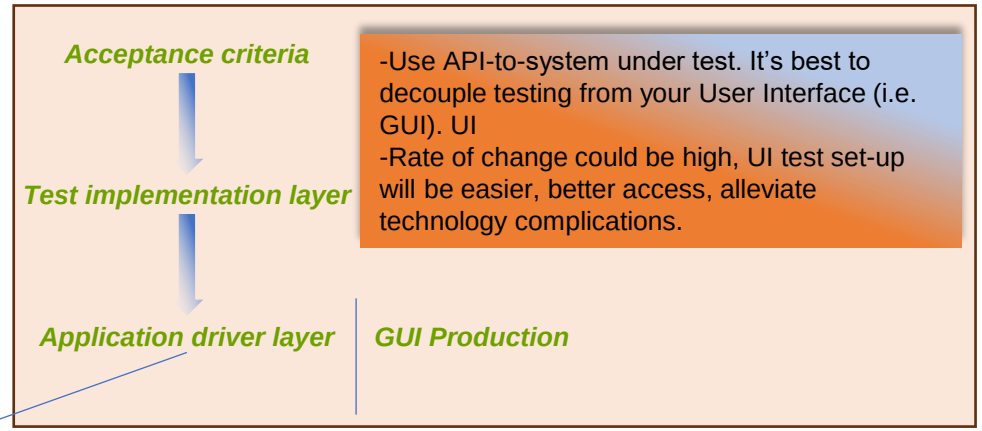
-Commit stage scripts should be modular

Tools & Technologies
Binaries: Jars, .NET assemblies, .so.
Outputs of commit (repository): Nexus, Maven.
Mocking tests: Mockito, Rhino, EasyMock, Jmock.

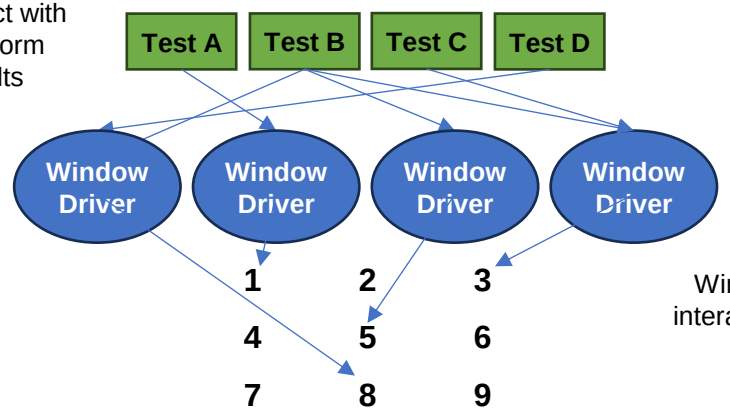
Automated Acceptance Testing

Business facing functional / non-functional tests that are automated against a running version of a production like environment…

Automated Acceptance Tests	
-1-unit or component tests 2-regressions tests. -$$$ but effective; lead to better software releases; help catch substantial errors.	-Automated tests are best for A-well factored apps B-thin UI layers C-loosely coupled architecture D-when there is alignment between developers, testers and end-users E-when they can run on Dev or PROD environments.
Note: Acceptance tests are executable specifications of the behavior of the software being developed. Acceptance tests – test what the application will do at any given point. Test the true value to the end-user (i.e., client).	
Acceptance criteria: 1-Given...2-When…3-Then >>> Behavior or feature-driven Feature-driven = set up users / personas; test features / behavior; isolate each scope; set up proper datasets.	Given = State or starting point (known-good); initial context When = Something happens; an event occurs Then = Is the result; there are outcomes…

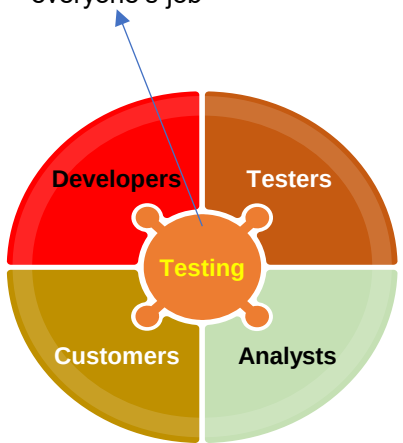


Understands how to interact with application *or* GUI to perform actions and return results



Window driver interacts with GUI

Testing becomes everyone's job



Implementing Acceptance Tests					
State	**Features**	**Asynchronous / Regression Testing**	**Test Doubles**	**Acceptance Tests**	**Testing (misc)**
-State is the condition or place of application (AKA stateful).	-User accounts or personas and make your tests specific to the scope of features. -Mimic the behavior of application during testing.	-Testing should appear as synchronous. -Increase quality through automated regression testing.	-Automated acceptance testing is not UAT. -Test doubles protect the external systems from being affected via simulation. -Decrease test time with parallel testing under a single instance of system.	-Should be run against every build that passes commit tests. -Keep acceptance tests green, free of any failures / breakages. -Free up expensive resources to focus on exploratory testing or other testing strategies.	-Fail early, fail fast; early feedback loops. -External user's viewpoint is in the test script. -More stateful acceptance testing; refactor common tasks in testing.

Tools & Technologies
Test automation tools: Twist, Cucumber, JBehave, Concordian.

Non-functional Requirements

NFRs are complex because they influence how the architecture of a system is built

Non-functional Requirements				
Performance	**Capacity**	**Automated Capacity Testing**	**Reliability / Usability**	**Scalability / Maintainability / Availability**
-Measure of time taken to process a single transaction. -Catch all term for capacity, throughput etc.	-Capacity is the max throughput for a given workload while maintaining an acceptable response time. -Throughput (e.g., # of transactions a system can process) -Scalability (e.g., response times and simultaneous users' effect on servers, threads and services) -Longevity (e.g., running the system for some time to see performance changes, degradation) -Load (e.g., # of concurrent users, log-ins)	-An automated capacity test suite should be created and run against every change to the system that passes the commit stage and (optionally) the acceptance test stage. Test real-world scenarios (e.g., not overly abstract). Have a pre-defined threshold for success for passed tests. Test in short durations. Address modifications early and fast. Be composable into larger-scale, complex and pragmatic scenarios. Make automated tests repeatable and enable sequential and parallel testing (e.g., suite of tests to apply load and run longevity tests).	-Requirements for NFRs should be clear (e.g., "The system shall return results in under 2 seconds" is too vague). -Don't create overly complicated testing requirements (e.g., capacity) too early in the process. It may be overkill.	-Use high-performance capacity testing that mimics production. -Load / stress during testing should scale to what production would look like. -Scale down tests. Example: One web server, two app servers, one database server.
	-Create capacity tests from your automated acceptance test suite. -Virtualization is not recommended for capacity testing.		-Incorporate non-functional tests into the deployment pipeline for best results.	

Capacity best practices…

Decide on architecture of app; pay attention to process / network boundaries / I/O.
Use patterns that enable stability/capacity of system.
Keep team working within boundaries of architecture; don't just try to optimize capacity.
Pay attention to data structures / algorithms and make sure they are applicable for the application.
Be careful about multi-threading (e.g., increase processor efficiency by dividing up work for simultaneous processing) for the core application
Establish automated tests that assert the desired level of capacity; fix the problems.
Use profiling tools as a focused attempt to fix problems identified by testing. No need to test the whole system, scale it down appropriately.
Use real world capacity measures (e.g., # of users; pattern behaviours; size of production data set.

Adding capacity tests to the deployment pipeline...
-Every change that passes the commit tests and acceptance tests should have automated capacity tests run against it.
-Capacity tests should be run in their own special environment because of timing. Other types of testing can be done in parallel – e.g., manual, integration etc. Capacity testing for (examples) 1-complexity 2-longevity 3-leaks 4-worst cases 5-integration failures 6-degradation 7-performance benchmarks 8-scalability 9-Third party application configuration 10-load testing with external systems.

Capacity Testing (Interactions, playbacks, recording etc)

User interface

-Not recommended for high volume, performance driven tests.
-Not recommended for distributed systems with complex client applications / centralized server-based components.

Public API

-Events, message queues, web services etc.
-Does not interact with client UI. Suited best for SOA environments.
-Use recorded interactions for testing.

Lower-level API

-Use lower-level API services.
-Make direct calls to a service layer or perhaps the database.

Pay attention to NFRs right from the start but don't develop defensive architecture or over engineer…execute automated tests for FRs and NFRs at every change commit.

Deploying & Releasing Applications

Powerful zero-downtime releases (with roll-back abilities) into production: Blue-Green deployments and Canary releases…

Deploying software to Testing environment

Releasing software into production

Deployment and Release Best Practices0

-Release products on demand and ensure continuous delivery of client-installed software.

-You will always know which version of the application is being deployed to what environment and by whom.

-This allows for quick roll-backs, as well as identification of failure points.

Release strategy considerations:

-Asset & config mgmt strategy; identify those in charge of deployment / release.

-Dev & Ops teams agree on technology.

-Environments for acceptance, capacity, integration and UAT.

-Deployment plan (e.g., runtime configuration, app deployment timeline etc)

-Monitoring (e.g., state), changes, integration with external systems (e.g., APIs), logging, disaster recovery planning.

-SLAs for failover or HA; production sizing and capacity planning.

-Archiving strategy, defect resolution, patches.

-Upgrades and application support in PROD.

Release plan:

-Plan for first (and subsequent) deployment; smoke test the application and services; back out plan; plan to back-up and restore the application's state; upgrades; logging / monitoring; data migrations; issue log.

-For legacy apps, plan to transfer users and decommission the old system.

Releasing products:

-For commercial use product releases…consider pricing, licensing, copyrights, marketing materials, product documentation, installers and sales support.

Deploying / promoting your application:

-Always create automated scripts no matter what deployment.

-A-Deployment pipeline commit stage B-PROD like readiness C-Automated process to take binaries to this environment D-Smoke test.

-"Virtualization" is key to mimicking PROD on a single machine. "Chicken-counting" means that you only need ~1-3 web servers not the actual 250.

Modeling your release process and promoting builds

-VSM of a release 1-Integration testing 2-Automated acceptance testing 3-UAT 4-Staging 5-PROD (operations).
-CI Release pipeline tools such as Go / AntHill Pro help you move through the releases (e.g., phase gates). Promote by button-pushing. Examples: Analysts / testers can do exploratory / usability testing; Ops folks can deploy any version to Staging or PROD.
-Workflow A-Each person on team should be able to deploy any versions of app that have passed all prior stages B-Infrastructure (e.g., middleware) should be in a clean state) C-Application's binaries are already available in a repository D-Application configuration is ready to go E-Data migration is prepared F-Smoke testing is ready to go G-Once testing is done, promote the app.

Promoting configuration

-Only configure what needs to be at any given time (e.g., don't promote to PROD a SIT database). Only promote what is application for PROD.
-For SOA / componentized apps, everything must be promoted together. All relevant components of the app must be promoted together.

Orchestration

1-Environments are often shared between apps. Use virtualization to isolate apps within environments.
2-Applications may depend on other apps (e.g., SOA apps); Use SIT to sync up the apps before moving to other environments.

Deployments to staging environments

1-Deploy to staging *prior* to PROD for any 1-kind of mildly complex systems 2-external sys integrations. Ensure that PROD, capacity testing and staging env are commissioned and ready 2-have auto configuration ready (e.g., networks, external services, infrastructure) 3-smoke test deployment process 4-measure warm-up period for app (e.g., caching) 5-test integration with external sys 6-get app into PROD well before official "release" (e.g., reconfigure router to redirect traffic from some holding page to PROD, blue-green) 7-roll out app to a small group of users (e.g., Canary releasing) 8-deploy every "passed" change to staging ASAP.

Rolling Back Deployments & Zero-Downtime Releases

Rolling back deployments

-Roll back to a previous good known version with data, integration with other systems and required reconfiguration in mind. The whole point is to be able to roll back to a good last version that is bug free.

Zero-downtime releases

- AKA hot deployment – decouple various parts of the release process (including shared resources (e.g., databases, servers etc) and allow for instantaneously switch users.
- **Blue/Green** – Keep 2 versions of PROD (e.g., web/application server, database) and be able to switch from green to blue in less than a second.
- Blue/Green can be 2 separate PRODs, Staging / PROD, 2 virtual instances, 2 databases etc.

Tools & Technologies

CI Releases: Go; AntHill Pro

Configuration management: Escape [apvrEr]

Configuration items monitoring (middleware): Nagios

Deploying & Releasing Applications

The most crucial part of release planning is assembling representatives from every part of your organization involved in delivery: build, infrastructure, and operations teams, development teams, testers, DBAs, and support personnel.

Rolling Back Deployments & Zero-Downtime Releases

Zero-downtime Releases

- **Canary releases** – roll out to a subset of users to one part of PROD. Roll out to new application to a new server, smoke / capacity test it, then route a subset of users to the new app.
- Canary releases are similar to A / B testing, where you can test capacity, CPU usage, I / O, memory, exceptions etc. You can roll back easily. Canary releases can be a challenge when there are shared resources (e.g., databases).
- They are helpful where there are a large, highly distributed systems with thousands of clients. This meant that different collections of stores would be operating with different versions of the client system, talking to different versions of the server-side system **but** all sharing a common underlying database.

Emergency Fixes

- Never subvert your process if there is a critical defect. You must go through the same build, deploy, test, and release process as any other change (e.g., making uncontrolled changes in PROD). This can lead to bad regressions and patches; also, the changes may not be recorded. You end up in an unknown state.
- For emergency fixes, use the standard deployment pipeline and configuration process.
- 1-Never do them late at night and always pair with others 2-Make sure to test the emergency fix process 3-Only circumvent the standard process in extreme situations 4-Test in Staging env 5-Sometimes it's better to roll back to a previous good version.

Continuous Deployment

-Loosely based on Extreme Programming (XP), the logical extreme is to deploy every change that passes your automated tests to PROD that passes all automated tests.

-Continuous deployments through canary releases ensure that new releases will work in PROD. High quality software + high quality process = fast, reliable deployments to PROD.

-**User or client-installed software:** 1-Manage the upgrades 2-Migrate binaries, data & config 3-Test the upgrade 4-Get any crash reports. You have to manage the large # of versions of the software.

-How clients can handle the upgrade process: 1-Have the software check for new versions and prompt the user to download and upgrade to latest version 2-Download in the background and prompt for installation silently (user will get prompt) 3-Download in the background and upgrade next time the app is started. Make it bullet proof and silent. Keep different versions of directories and simply rename to the correct version once everything is working. Run scripts to roll forward versions with correct data stores and config.

Tips & Tricks for Continuous Deployment

People Who Do the Deployment Should Be Involved in Creating the Deployment Process

-Dev & Ops teams must be involved in the process.
-Example: Members of deployment team work with dev team to create scripts + installation docs on a wiki. Ops team became confident in their abilities to deploy quality software because of these sessions.

Log Deployment Activities

-Keep a log of all files that your automated deployments process copies or creates. Then, it will become easy to debug problems (e.g., config, logs, binaries, provisioning of env).
-Keep log of all hardware, which bits you touched during deployment, and logs of actual deployments.

Don't Delete the Old Files, Move Them

-When you do a deployment, keep a copy of the previous version. Keep stray files out of any new deployment versions (e.g., old interface page).
-Example: In the UNIX work, use a symbolic link that points to the current version and mark a symbol for the previous link. For networks, keep different versions as well or different port ranges on the same server.

Deployment Is Whole Team's Responsibility

-A "build & deployment expert" is an antipattern. Every member of the team should know how to maintain deployment scripts. Build software by always using real scripts.
-A broken deployment script will break the build.

Server Applications Should Not Have GUIs

-PowerBuilder and Visual Basic used to have GUIs on their server apps (e.g., configs were not scriptable, apps installation issues). A user had had to show the UI for the build to work.
-PowerBuilder 2003 example: A trading company used PowerBuilder. The app had a GUI that had to be started everyday causing processing errors. There were also delays with an old version of Windows.

Have Warm-Up Period for a New Deployment

-With canary releasing, you can "warm up" your new servers and releases can start by serving small portions of requests. Add more load slowly.
-In a production-like environment, test how long it takes for your cache to fill up at deployment time. This will help gauge anticipated response times.

Fail Fast & Fail Early

-Use simple, non-comprehensive smoke tests to ensure deployment was successful.
-The system should perform these checks as it initializes and, if an error occurs, it should fail to even start.

Don't Make Changes Directly on the Production Environment

-Only your deployment pipeline can make changes to PROD.
Use limited-lifetime passwords created by an approval process + two-phase authentication. Example: In one org, changes to PROD could only be authorized from a terminal in a locked room with a CCTV camera.
-Have an audit trail for everything going into PROD.

Managing Infrastructure & Environments

The Environment

-1-Hardware configuration of the servers that form the environment (e.g., # of CPUs, memory, spindles, NICs + networking infrastructure 2-Config of O/S + middleware (e.g., messaging systems, application/web/database servers) for app support.

-Boundaries between app and infrastructure go across embedded software with distinct infrastructure to SOA where infrastructure is shared across apps.

-A-Desired state of infra should be specific through version control B-infra should be autonomic C-Know state of your infra through monitoring.

-Manage these things carefully: O/S config, middleware stack config, version control, directory services, external integrations, network infra (e.g., routers, switches, DNS). App & infra teams must in sync.

Operations (within DevOps)

- Dev teams care about delivering software as quickly as possible. Ops teams aim for stable products. Collective goals is to build safe products by releasing software continuously.
- Ops teams focus on Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR). Ops teams have SLAs.
- Concerns for Ops teams:

Documentation & Auditing: This includes change control process (e.g., ITIL/ITSM). Dev teams should have an understanding of SOPs required by Ops teams (e.g., release plan).

Alerts for Abnormal Events: Ops managers will monitor running infra/apps. Use tools such as OpenNMS to monitor/log events. In case of failures, Ops teams should be able to redeploy. Every new app has its own set of issues.

IT Service Continuity Planning: Ops managers will be involved in IT service continuity planning (e.g., Recovery Point Objective (RPO) a measure of the length of time prior to a disaster for which data loss is acceptable, and a Recovery Time Objective (RTO) the maximum length of time allowed before services are restored. You must be able to pull back to previous versions. For HA apps, you must replicate data/config live. Production data must be archivable.

Use Technology Ops Team Is Familiar With: Ops teams might be familiar with Bash or PowerShell but not Java/C#. Ops teams, none-the-less, must be familiar with env/infra being deployed.

Agreed upon technology at the beginning of the project is essential between Dev & Ops teams. Use a scripting language (e.g., Perl, Ruby, Python, or packaged tech such as Debian or WiX. Both Dev & Ops teams must understand the full deployment process with a release plan (e.g., deployment version control, proper scripts).

Infrastructure

Configuration management implementation is complex. To automate your build, integration, testing and deployment, you must answer 1-How to provision infra 2-How to deploy & config various bits of software that form your infra 3-How to manage your infra once it is provisioned/configured. Example: For complex, multi-user apps, default settings of O/Ss & middleware will not work. You will need to configure access control, firewalls etc. And, you will need to set up databases, permissions set up, application server components deployed, defined message brokers etc. Keep everything under version control.

Keep everything you need to create and maintain your infra under version control: **O/S Install Definitions:** Debian Preseed, RedHat Kickstart, Solaris Jumpstart. **Config for Data Center Automation:** Puppet or CfEngine. **General Infrastructure Config:** DNS zone files, DHCP / SMTP server config files, firewall config files. **Any Scripts for Managing Infrastructure.**

-These files in version control form inputs to the deployment pipeline the same way source code does. Job of deployment pipeline in case of infra changes A-verify that all apps will work with any infra changes prior to PROD release including apps' functional/non-functional specs B-should be used to push changes out to Ops managed TEST/PROD env C-pipeline should perform deployment tests to ensure new infra config has been deployed successfully.

Note: Infra config for one particular app should be part of the deployment pipeline with no separate lifecycle. Infra, if shared between apps, involved dependencies. You must use the correct infra version for each app it depends on and set up a separate pipeline to push out infra changes to each app's delivery.

Tools & Technologies

Ops Monitoring: OpenNMS, Nagios, HP's Operations Manager.

Tools for Ops Teams: Bash, PowerShell.

Common Scripting Languages for Dev & Ops: Perl, Ruby, Python, or packaged tech such as Debian or WiX.

O/S Install Definitions: Debian Preseed, RedHat Kickstart, Solaris Jumpstart.

Config for Data Center Automation: Puppet or CfEngine.

Managing Infrastructure & Environments

Infrastructure…

Access & Changes

Access: 1-No change without approval 2-Automate changes 3-Monitoring for issues.

-Make changes at set times with change control process. Restrict both Dev & Ops folks as to who can make changes.

Changes: 1-Same change process each time 2-Single ticket view for everyone 3-Logging for audit 4-History 5-Automated tests on PROD like env 6-Verify success.

Managing Server Provisioning & Configuration

Provisioning

-Automated server config & provisioning install, usually remote (i.e., virtualization).

Management

-A-You have a consist env B-You can easily provision new env C-You can put in a new box if there is hardware failure. See Tools & Tech for infra mgmt across Microsoft, UNIX. All of these tools get you to the desired end-state.

-Puppet is one of the most popular for config (e.g., propagates change to all clients). Add simple lines to Puppet Master's definitions and you can roll out new env all over the world.

Managing Config of Middleware

-Examples: Database schemas, web server config files, app server config info, message queue config…

-Usually, middleware can be managed along with your O/S using Puppet.

-Example of good middleware config: Put app server's installation directory in source control, write a script to copy it to the correct env, put the app server's config file for the env in its directory and run automated scripts for deployment.

-Open-source systems / components can provide best scripts for config. Automation is key for successful config of middleware, but you can do it manually too through XML scripts or even through databases. You can also use configuration APIs

Managing Infrastructure Services

-It is very common for infrastructure services (e.g., routers configs, DNS, firewalls, directory services) to break software in PROD. A-Version control everything B-Monitor everything C-Utilizing logging C-Smoke test all of these D-Make integration in TEST mimic PROD. See T&T.

-Harden multihomed systems and multiple, isolated networks for varied traffic.

Virtualization

-Platform virtualization refers to simulating an entire computer system so as to run multiple instances of O/Ss simultaneously on a single physical machine.

-Virtualization reduces time to deploy software (e.g., server consolidation).

-**Benefits:** Fast response to changing requirements, consolidate CI servers & testing infra, standardizing hardware to be able to run heterogenous envs, easier to maintain baselines using server images from a library.

-Virtualization can create new envs at the push of a button. Virtualization can replace automation when not avail. Easy to use "known good version."

-**Virtual envs & deployment:** Virtualization can help with deploying changes to PROD (e.g., bug fix, change to source code etc). Use of virtualization templates helps move changes to PROD.

-**Virtualization for parallel testing:** Virtualization provides an excellent way to handle multiplatform testing (UNIT, TEST, PROD).

Cloud Computing

-Three categories of cloud computing: apps in the cloud (Salesforce, Gmail) platforms in the cloud and infra in the cloud (AWS, GCP, MA). Example: AWS provides message queues, static content hosting, streaming video hosting, load balancing, storage.

-Security and SLAs are very important. Example: Healthcare company TC3 encrypted its data in order to be able to host its service on AWS for HIPAA compliance. Example: If you need to run a very large dataset with a high load, you may not want to put it in a virtualized env.

-**Platforms in the cloud:** Google App Engine, Force.com platforms providers take care of things like scaling the app and infra. The provider takes care of non-functional reqs (e.g., scalability, avail, security).

-Cloud computing can take care of the full deployment pipeline in seconds (deploy, test, release). But, platforms such as Google App Engine has constraints on what it can provide (e.g., types of databases). Also, vendor lock in also an issue since you have to choose between AWS, GCP, MA plus public, private or hybrid cloud.

-Cloud computing uses loosely coupled architecture.

Monitoring Infra & Apps

-Monitoring is great for 1-real-time intelligence (e.g., revenue) 2-incident response timing 3-historical data collection for planning. Areas to consider are A-making your app/infra ready for data collection B-storing the data for analysis C-dashboarding D-notifications to employees for events relevant to them.

-**Collecting data:** from hardware, O/S, middleware, apps.

-**Logging:** used for patterns and behavior; log issues. Used for audit.

-**Dashboards:** help with analytics and real-time tracking of risks and impact.

-**Behavior-driven monitoring:** similar to behavior-driven development. You can set up a scenario to test whether your config worked correctly.

Tools & Technologies

Test automation of changes: Puppet, CfEngine, BladeLogic, Tivoli, HP Ops Center. Selenium Grid for parallel testing.

Server config / provisioning: PXE <> Puppet.

Infra mgmt: Microsoft (System Cetner config Manager); UNIX access controls; BMC (BladeLogic); IBM (Tivoli); HP (Ops Center).

O/S config mgmt: CfEngine, Puppet, Chef. Example: Puppet can be used for version control as well.

Automated config of middleware: XML scripts.

API config: Microsoft's IIS using XML metabase or PowerShell.

Monitoring/Data Collection: Nagios, OpenNMS, HP Ops Mgr. Also, Flapjack, Zenoss, IBM Tivoli, Splunk.

Forensic tools to fix infra issues: Wireshark and Tcpdump. Lsof for UNIX. TCPView and Handle.

Managing Data

Applications…

Managing Data

Data Scripting:

-Any changes to your database should be managed through the automated process via database scripting using version control. Scripts should be part of your continuous integration process.

1-Initializing databases: create a script for database initialization that creates a structure of the database, database instances, schemas etc using version control. A-erase what was there before B-create a new database structure, instances, schemas etc C-load database with data.

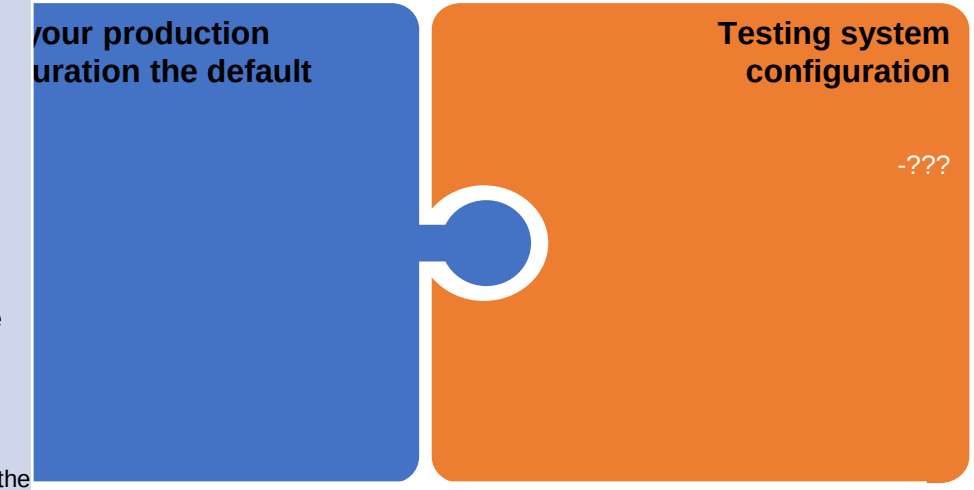
Incremental Change:

-Continuous integration demands that we the app working after every change, including structure/content of data. Update your operational database before any change goes into PROD.

Versioning your database: versioning your database using a table. Add scripts for version control (e.g., roll forward/backward scripts). See T&T.

-Point is to be able to continuously deploy your app with the most update data and allow your DBA to make changes to data for migration without affecting the app.

Managing orchestrated changes: one way to manage incremental changes is to make apps work with multiple versions of your database, so that the database can be migrated independently of the applications it depends on (i.e., zero-downtime releases/deployments). This way, the database operates independent of the apps that depend on it. You can ensure that the latest version of the data is being migrated as part of the deployment.



gement (CM) visibility – Managing configurations across applications

Tools & Technologies

Roll-forward/backward scripts for data: Ruby on Rails (ActiveRecord), Tarantino, Microsoft's DbDiff, Ibatis' Dbmigrate.

CI / CD Best Practices…

Continuous Integration and Continuous Deployment (CI/CD) are not just best practices but essential for developing and deploying robust GenAI models. GenAI models promise immense promise, but the path to effective deployment is complex. This is where CI/CD comes into play, offering a streamlined, automated approach to integrating and deploying these sophisticated models.

Implementing CI/CD will provide your organization with faster time to market, allow you to respond quickly to clients' demands, and launch new models at the speed of innovation.

First, you need to consider some architecture patterns for CI/CD, such as using microservices within an event-driven architecture where services communicate through events. Changes in one part of the system can automatically trigger actions elsewhere, ensuring real-time responsiveness.

Secondly, you need the right tools. Jenkins, an open-source automation server, is a reliable choice that supports building, deploying, and automating projects. GitLab CI/CD provides powerful capabilities right out of the box, giving you the confidence that your project is in good hands. Kubeflow simplifies deploying ML workflows on Kubernetes, ensuring a smooth / efficient process. MLflow manages the lifecycle, from experimentation to deployment, providing a seamless process.

Finally, embracing an MLOps framework is indispensable. Start with version control for data and models, just as you would with code. Automated testing and validation integrated into your CI/CD pipeline mean models are rigorously tested for accuracy, performance, and bias before deployment. Continuous monitoring of deployed models is essential to detect performance drifts and maintain effectiveness over time.

CI/CD and MLOps are not just technical necessities—they're strategic advantages that reduce time to market and deploy hi gh-quality, reliable AI models. By implementing CI/CD and MLOps, you are not just following a process, but you are actively contributing to the success of your organization and the advancement of AI technology.

DevSecOps – Consolidated Tools & Technologies

DevSecOps – Processes & products

DevSecOps	Product / Tool / Technology	Notes
Development / project management	Jira (Atlassian); Confluence; Jazz, ServiceNow	Need unified tool. One backlog. Most popular. Confluence can be bought with Jira.
Web application	Node.js, React.js, C#, Microsoft .net, .net Core, Angular.js, Java/J2EE, LAMP	
Mobile	Xamarin, Objective C, Android, React Native, Ionic, Cordova, Flutter	
Operations	ServiceNow	Most used.
Programming languages	C++, Java, Python, Ruby	
Communication mechanisms / Chat rooms	Slack, Hipchat (Atlassian), Openfire (Apache)	
Automated code quality/review	SonarQube	Inspection of code quality to perform automatic reviews with static analysis of code to detect bugs and code smells on 29 programming languages.
Infrastructure-as-Code (IaC)	Terraform, Ansible	Terraform - It is a tool for building, changing, and managing infrastructure in a safe, repeatable way. Ansible - Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.
Cloud	Amazon Web Services (AWS), Google Cloud Provider (GCP), Microsoft Azure	
Microservices / Containers	Docker, Kubernetes, HELM	
Core repositories / version control	Git, GitHub	
CI/CD Pipeline	Jenkins, Git, GitHub {Actions}, GitLab CI, Bit Bucket	
Mocking tests	Mockito, Rhino, EasyMock, Jmock	
Monitoring / event logging / observability	-Event management: Splunk, ELK stack (Elasticsearch, Logstash, Kibana). -Monitoring & logging: Grafana, Prometheus, SolarWinds,New Relic, AWS CloudWatch, AppDynamics, ELK stack, Oracle Enterprise Management (OEM), Microsoft SCOM (System Center Ops Manager).	

DevSecOps – Consolidated Tools & Technologies

DevSecOps – Processes & products

DevSecOps	Product / Tool / Technology	Notes
Configuration storage	YAML in Ruby, JAVA, XML, Restful Web Server	
Application configuration monitoring	Nagios, OpenNMS and HP OpenView	
Security	Prometheus, Grafana, SolarWinds	
Chat rooms	Slack, Hipchat, Openfire	
Virtualization / Configuration Management	Puppet, Chef	
O/S configuration automation	Puppet, CF Engine	
Code quality	Simian, JDepend (Java), NDepend, CheckStyle, FindBugs	
Automation testing tools	Cucumber, JBehave, Twist, Concordian, Selenium (Apache license 2.0)	
CI Servers	Go, Team City, Atlassian’s Bamboo, Zutubi’s Pulse	
CI tools	Hudson, Cruise Control	
Commit	Nexus, Maven	

CI / CD Best Practices…

- Continuous Integration and Continuous Deployment (CI/CD) are not just best practices but essential for developing and deploying robust GenAI models. GenAI models promise immense promise, but the path to effective deployment is complex. This is where CI/CD comes into play, offering a streamlined, automated approach to integrating and deploying these sophisticated models.

- Implementing CI/CD will provide your organization with faster time to market, allow you to respond quickly to clients' demands, and launch new models at the speed of innovation.

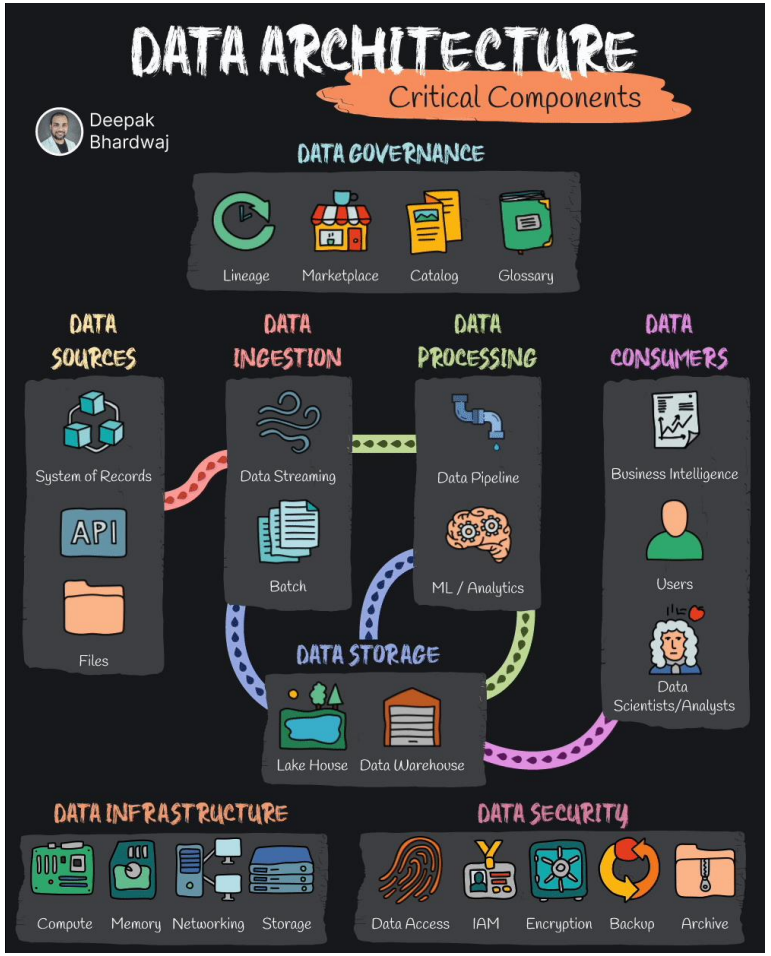
- First, you need to consider some architecture patterns for CI/CD, such as using microservices within an event-driven architecture where services communicate through events. Changes in one part of the system can automatically trigger actions elsewhere, ensuring real-time responsiveness.

- Secondly, you need the right tools. Jenkins, an open-source automation server, is a reliable choice that supports building, deploying, and automating any project. GitLab CI/CD, integrated directly with GitLab, provides powerful capabilities right out of the box, giving you the confidence that your project is in good hands. Kubeflow simplifies deploying machine learning workflows on Kubernetes, ensuring a smooth and efficient process. MLflow manages the entire lifecycle, from experimentation to deployment, providing a seamless process you can trust.

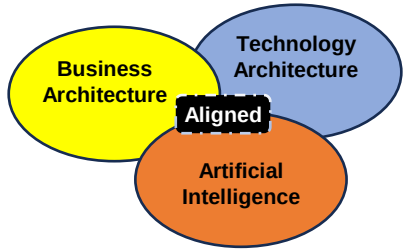
- Finally, embracing an MLOps framework is indispensable. Start with version control for data and models, just as you would with code. Automated testing and validation integrated into your CI/CD pipeline mean models are rigorously tested for accuracy, performance, and bias before deployment. Continuous monitoring of deployed models is essential to detect performance drifts and maintain effectiveness over time.

- CI/CD and MLOps are not just technical necessities—they're strategic advantages that reduce time to market and deploy high-quality, reliable AI models. By implementing CI/CD and MLOps, you are not just following a process, but you are actively contributing to the success of your organization and the advancement of AI technology.

Data Architecture



Artificial Intelligence – Digital Transformation



Blue Ocean Thinking

- Artificial Intelligence (AI) will not replace business architecture; rather, it will complement it and offer a new avenue to simplify complexity and enhance technology. AI will position companies for success and differentiated technology enablement.
- AI has many components; some of the more notable ones are Machine Learning (ML), Intelligent Automation (IA), Robotic Process Automation (RPA), and Blockchain. Think big, execute small/Agile, integrate continuously.

Artificial Intelligence (AI)

- AI is the concept of computer systems intelligence that can perform human tasks exponentially faster than humans, and with better accuracy – e.g., visual perception, speech recognition, decision-making, language translation, ambient dictation etc. Moreover, for AI to really work, computer systems learn and adapt on their own. Example: a computer system learns the basics of poker but then learns to become an expert on its own. The problem with IBM Watson was that it was an input/output system, where you had to feed it millions/billions of data points to process information. It didn't evolve on its own like AI.

Machine Learning (primary AI)

- ML (and Deep Learning) is a largest sub-discipline of AI. ML uses high quality data and is trained to apply algorithms to solve problems.
- As a business use case example, for a loan application, ML can be applied to approve / deny a client's loan request. This removes human bias (e.g., approving loans for men *over* women or whites *over* blacks.

A-Data is critical because it trains the ML Models which are the core of decision making and they drive the Application(s). Therefore, the data must be high quality for optimal Business Processes which are fed by the application(s). B-Business domain knowledge must be built into the process architecture for workflow automation by priority. Example: Automate a process architecture that gives you competitive advantage. C-Software such as Cloud, CRM, ERP, EMR, Microsoft Office etc systems must be updated to handle ML. D-Ensure that transformation can be done systematically using an EA framework (e.g., global optimization) E-ML is a probabilistic approach not just "if / then" statements F-Leadership must take a holistic view to infuse AI/ML.

Intelligent Automation (IA)

- IA is general term for automating routine, manual tasks such as front-end registration and scheduling in healthcare or claims processing on the back-end. IA boosts productivity, exponentially speeds up task completion, improves accuracy and reduces costs. IA complements AI where IA is computer intelligence that mimics human performance while AI teaches computer systems to think on their own (i.e., autonomous learning, automation, predictive/prescriptive analysis etc).

Robotics Process Automation (RPA)

- RPA is a type of IA that uses bots to perform repetitive tasks that humans typically execute. RPA is simply a computer system that performs millions of manual tasks automatically and intelligently. RPA, like AI, has a way of becoming more intelligent over time. RPA is very popular in healthcare for both front-end and back-end processes. For example, RPA can be used to accurately process, modify, and adjudicate insurance claims.

Generative AI

- The latest hype, Gen AI creates new content such as text, audio and video, with applications in chatbots and deep fakes. For example, Gen AI can be used for ambient digital scribing from patient <> dr conversations.

Blockchain

- Blockchain is a decentralized, distributed ledger with a list of records (e.g., blocks) that are individual transactions linked together via cryptographic hashes. When most people think of blockchain, they think of Bitcoin or other cryptocurrencies. However, blockchain is beyond crypto and can be used in healthcare for patient data management, insurance claims processing, data interoperability, payment processing, pharma supply chain etc. Blockchain is decentralized, making it much harder to steal data. No one seems to talk about blockchain much anymore in 2024.

Artificial Intelligence – Healthcare Best Practices

These are just examples. There are numerous other ways AI can improve healthcare for and within payers, providers, life sciences, bio, pharma etc.

Artificial Intelligence (AI) + Generative AI*

Illustrative

Healthcare Providers

1-AI-driven diagnostic tools have the ability to analyze medical images with exceptional precision, assisting doctors in the early detection and diagnosis of diseases. AI can analyze medical imaging data, such as X-rays, MRIs and CT scans, to assist healthcare professionals in accurate and swift diagnoses.

AI is helping doctors analyze images more quickly and effectively, seeking signs of breast cancer, lung nodules and many other conditions to reach more people with early detection than has previously been possible.

2-AI can improve utilization management, helping both payers and providers optimize healthcare usage.

3-AI can drive payment accuracy & integrity resulting in ~10% savings in annual spend.

4-AI can help with savings costs opportunities before medical services are performed.

5-AI has the ability improve clinical quality care, including sharing data & insights to help plans improve clinical quality, care access, health equity and member experience.

6-Gen AI - Ambient digital scribing is a perfect example of Gen AI. Gen AI can scribe patient-to-doctor conversations, clean up clinician documentation and notes etc.

New generative AI applications can extract data from patients' medical records, populate it instantly into forms, record notes from patient sessions and speed / improve patient communications.

7-AI can exponentially improve interoperability and value-based care between payers & providers. It can create transparency and collaboration, especially when it comes to data exchange, payment models, claims processing etc.

Health Payers

1-AI can improve utilization management, helping both payers and providers optimize healthcare usage.

2-AI can drive payment accuracy & integrity resulting in ~10% savings in annual spend.

3-AI can improve the claims adjudication process improving auto adjudication rates and reducing waste and admin costs.

4-AI can help with savings costs opportunities before medical services are performed. Note: it is essential for both payers & providers to track savings costs *before* any services are performed.

5-AI can analyze risk adjustment for payers as well, helping health plans secure adequate funding to administer member policies. This is essential for retrospective & prospective risk adjustment.

6-AI can exponentially improve interoperability and value-based care between payers & providers. It can create transparency and collaboration, especially when it comes to data exchange, payment models, claims processing etc.

Life Sciences, BioPharma & Medical Devices

1-AI is streamlining drug development with its ability to match drug targets with new molecules that can treat and cure diseases, saving time and money resulting in cheaper, better care for patients.

AI can help get new drugs to market faster and can reduce production costs, improve product quality and increase manufacturing throughput, ultimately accelerating the production and delivery of drugs to market.

2-AI can get medical devices to market faster by driving manufacturing efficiencies and reducing the risk of errors through Machine Learning (ML).

AI can also use medical devices to improve operational costs through tracking wait times and reducing inpatient length of stay in hospitals. AI medical devices' innovation can also help staff with diagnosing abnormalities via medical imaging from various medical devices. Note: each year, 3.6B medical images are produced in hospitals worldwide.

**Generative AI represents an innovative frontier in the field of artificial intelligence, focusing on the creation of new content, such as text, audio, images and video. At the heart of generative AI lie large language models, which are trained on vast amounts of data to predict and generate coherent sentences, paragraphs, and even entire documents.*

This technology has found widespread application in various domains, from the development of intelligent chatbots that can engage in human-like conversations to the creation of realistic deep fakes, which are synthetic media that closely resemble authentic content.

Artificial Intelligence – Blue Shield of Michigan (2024)

Blue Shield of Michigan has created a remarkable AI strategy and has quietly become a leader in innovative AI usage within its workflows…

Blue Cross Blue Shield Association

Blue Cross, Blue Shield of Michigan (BCBSM) is a $36 billion healthcare payer with more than 5 million members and a network of more than 37,000 providers.

https://hbr.org/2024/08/how-one-major-healthcare-firm-became-the-leader-in-innovative-ai-use

Response to Crowdstrike (July 2024) Issue

1) Ensure software updates undergo extensive testing in diverse environments before deployment. Simulate real-world scenarios to identify potential issues early. Implement staged rollouts, initially deploying updates to a small subset of systems to monitor and address issues before a full-scale release.

2) Robust Contingency Planning: Develop and maintain a detailed incident response plan covering identification, containment, eradication, and recovery. Regularly update business continuity plans to mitigate the impact of IT disruptions on critical services.

3) Communication Protocols: Establish clear communication channels with clients and stakeholders to inform them of issues and provide regular updates. Ensure effective internal communication among IT, security, and operations teams for swift, coordinated responses.

4) Access Control and Privilege Management: Apply the principle of least privilege to software updates to minimize potential damage from faults. Use network segmentation to isolate critical systems and limit the spread of disruptions.

5) Monitoring and Logging: Implement robust real-time monitoring systems to detect and respond to anomalies quickly. Maintain detailed logs of all activities related to software updates and system changes for post-incident analysis and auditing.

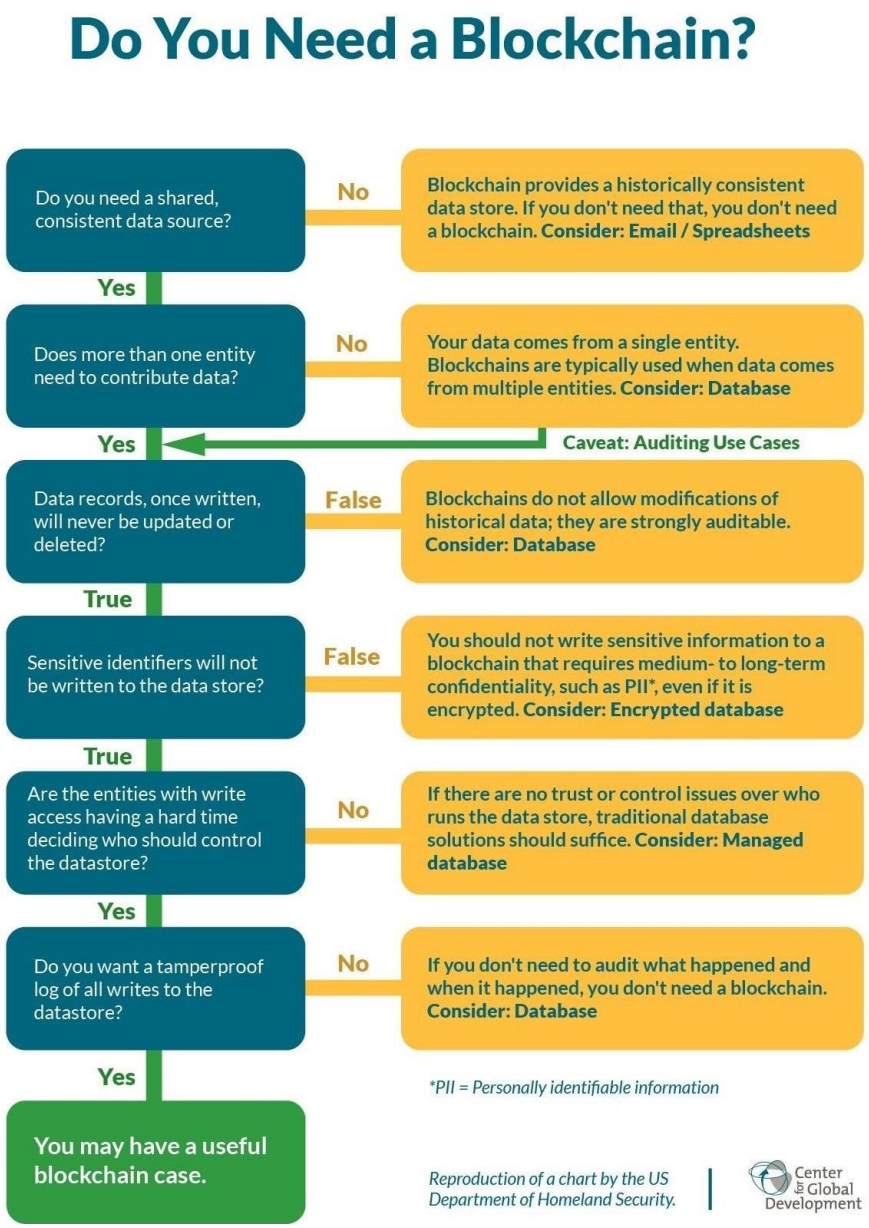
6) Vendor Management: Regularly evaluate and audit third-party vendors to ensure adherence to high security and quality standards. Establish clear SLAs with vendors, including clauses for timely incident response and resolution.

7) Training and Awareness: Conduct regular training sessions on cybersecurity best practices, emphasizing adherence to update protocols and issue reporting. Perform regular simulation drills to test the effectiveness of incident response and recovery plans.

8) Update and Patch Management: Utilize automated tools for update and patch management to streamline processes and reduce human error. Ensure well-documented rollback procedures are in place to quickly revert to previous stable versions if issues arise.

From someone on LinkedIn…

Do you need a Blockchain?



DevOps Handbook

Product, Project & Program Management

DevSecOps – Consolidated Tools & Technologies

	Project Management	Product Management	Program Management
Focus	You manage specific projects from start-to-finish. You are brought on for a specific purpose and goal.	You manage the lifecycle of the product. You raise the product through its life like a baby. It is your baby.	You manage a group or portfolio of projects. You may manage several workstreams on a large-scale implementation.
Timeline / Lifecycle	Your timeline is project-specific and could have a wide range (e.g., 3-year ERP implementation, 6-week IT infrastructure assessment. You have a finite timeline.	Your timeline is on-going. It never ends. You grow with the product through its evolution.	Your timeline could be longer on the program than it is for project managers. You may oversee several projects for a longer period.
End Goals	You deliver the project with the big 3 in mind 1-Scope 2-Budget and 3-Timeline.	Your focus is to deliver a successful product that meets user needs. You manage each successful version deployment (e.g., v1.1, v1.2, v1.2.1)	You deliver the project with the big 3 in mind 1-Scope 2-Budget and 3-Timeline. You may act as the Program Management Office (PMO) for a large program.
Responsibilities	You plan, execute and close projects. You follow a project plan. The product owner manages the backlog of implementation.	You define the product vision, strategy and roadmap. The product owner manages the backlog of implementation.	You oversee, often as a PMO, the planning, execution and closing of several projects.
Stakeholders	Your stakeholders are the project team, clients, upper management and end-users.	Your stakeholders are end-users / customers, development team, marketing / sales teams, clients and upper management.	Your stakeholders are the project team, clients, upper management and end-users.
Decision-making	Decisions / adjustments are made based on project constraints (e.g., scope, time, budget).	Decisions / adjustments are made based on market research, user feedback, competitive analysis and business needs.	Decisions / adjustments are made based on program constraints (e.g., scope, time, budget).
Team / Organizational Structure	Cross-functional but everything is done with project end goals in mind.	Highly cross functional involving multiple departments such as design / development, marketing / sales etc.	Cross-functional but everything is done with program end goals in mind.
Risk Management	You identify and mitigate risks related to the project, one risk at a time.	You manage risks related to the product's market fit, competition and changing landscape.	You identify and mitigate risks related to the program, one risk at a time. You help manage each project's risks.
Success Metrics	Success is measured by project completion, time, scope, quality etc.	Success is measured by product adoption, usability, end-user satisfaction, revenue / project generation.	Success is measured by program completion, time, scope, quality etc.
Sample Deliverables	Successful EMR implementation for a healthcare client. Go-live was successful within the 10-month timeline.	Successful launch of a Health Information Exchange (HIE) product that will be vastly enhanced over 5 years.	Successful implementation of a complex healthcare ERP platform utilizing Oracle technology modules.
Career Outlook	You can have a very successful career in health tech moving from project to project.	You will enjoy being a product manager as it is a highly desired by most young leaders. You get to oversee the evolution of one product over years. You grow with it.	Most project managers become program managers given the additional authority and autonomy to manage a portfolio of projects.