

Comprehensive Case Study : Phase D

Files of this Case Study are to be referred and continued only in this sequence order, please :

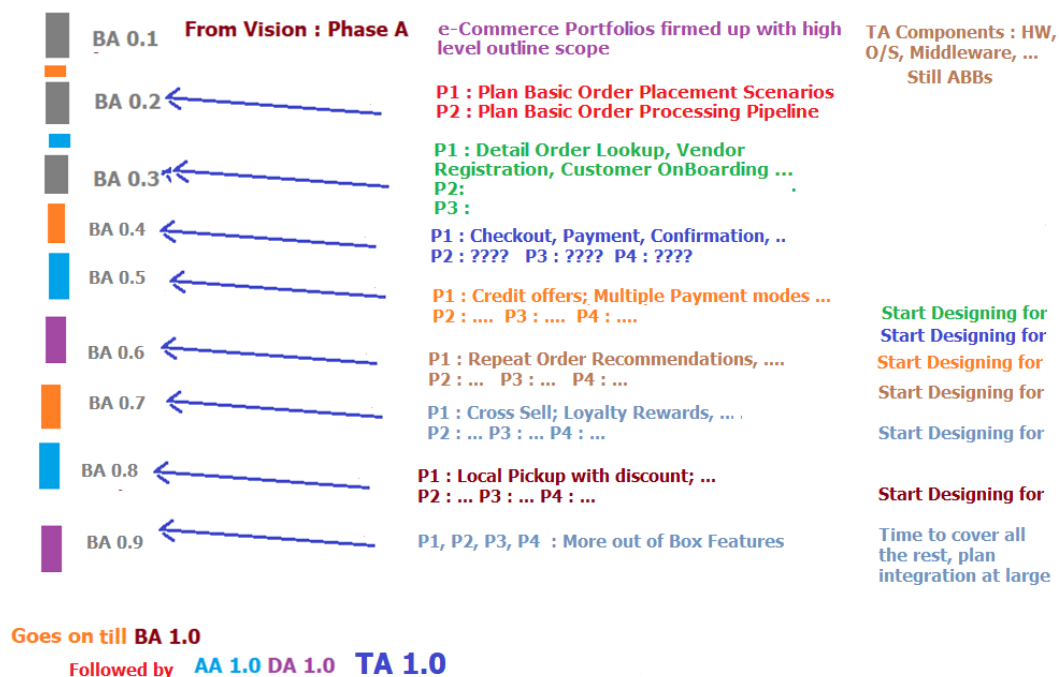
-  CompCaseStudyStart
-  CompCasePhA
-  CompCasePhB
-  CompCasePhCAppArch
-  CompCasePhCDataArch
-  CompCasePhD
-  CompCasePhE
-  CompCasePhF
-  CompCasePhG
-  CompCasePhH

Here, an EA takes up the Phase D – Technology Architecture work of the Portfolio

The Technology Solution Architect will be working under the guidance of the EA.

Note that in TOGAF, Technology Architecture is more than just Infrastructure. It is about all platform and network and connectivity aspects, which are all needed to execute the Application and Data Executables. Could be on Cloud or otherwise.

Do not forget that work progresses in Version Slices : Something like :



Points of Essence : Within Steps of Phase D : Technology Architecture :

Select Viewpoints

Get ready to meet Stakeholders and think in terms of Technical Context
furthering the Value Chain built through Business Architecture
and the Logical path build through Application and Data Architectures

Viewpoint from needs of Implementation Platforms & technologies;
Framework & runtime tools

Viewpoint from various Infrastructures : Software platforms, Hardware,
Connectivity ...

Viewpoint from Systems and IT Operations Management, Lifecycles and
many NFrs

Select Reference Models, Tools

Where to look for more guidance (from our Repository), What to refer
(rich library of Reference Materials – TRM,)

Hosting and Cloud Patterns
and what Tool / techniques to use in producing the target result

UML Tools and Other

What is already available as developed Baseline Architecture

Previous project BA, AA, DA and TA of completed versions, artifacts,
current project previous works :

Diagrams, Structures, Execution descriptions

White papers, POC Infrastructure segments

What to develop as Target Architecture

Not entire artifact list of TOGAF; a practical and useful set of Building Blocks suggested

Some BBs may focus on Enterprise-wide Technology Architecture rather than being Portfolio Specific

Do not forget that infrastructure is shared across applications and data systems

Also note that Cloud poses a change in Infrastructure approach

Technology Segment Architect finds the exact Gap of what is needed, performs by producing Building Blocks



And so on, looking into Cloud enablement where appropriate, interoperability with all existing infrastructures where needed, considering the latest technology in affordable and subscriber solutions

While proceeding in each version slice, covered many Data-level gaps and came out with Building Blocks : Consistency, Accuracy, Completeness based gaps, Transformational and Translation Gaps, Data Ingestion Automation gaps, Movement of Data objects and gaps therein, Concurrency areas, Sync and Async communication with the data stores and event-sourced updates

Building Blocks including Data structures and relationships

Class structures and data access relationships

Fit all work pieces in the defined Business Roadmap

Strategic, Tactical, Continuous change landscape with integration at
infrastructure level

To be in tune with long-term Enterprise Landscape

Resolve Impacts : Across Architecture Landscape. From TA angle

Resolution sessions with Business, Application, Data Segments

Is this ok to fit as one or more engineering
related segments and components ?

Are these in tune with Business level Views and artifacts ?

Is this information need ok to meet application and Data Component
demands ?

Stakeholder Review of these initiatives

Review relevant portions with relevant Stakeholders : as per Stakeholder
Matrix

NFRs need focus and discussion with many discrete stakeholders

Agreed ? : Great

Want modification ? : discuss pros and cons

Trade-off between Stakeholders needed ? : escalate through NFR, AOP
considerations

Finalized Technology Architecture

At end of every version slice, that much Architecture is finalized and documents (ADD, ARS) updated

EA decides if it is ready for reuse by others (Landscape placement in repository) or not

Select (TA) Viewpoints, (TA) Reference Models, (TA) Tools

A Step in Technology Architecture

Viewpoints : What points of view, of which Stakeholder to focus and proceed during TA work ?

Context of Tools and Environment

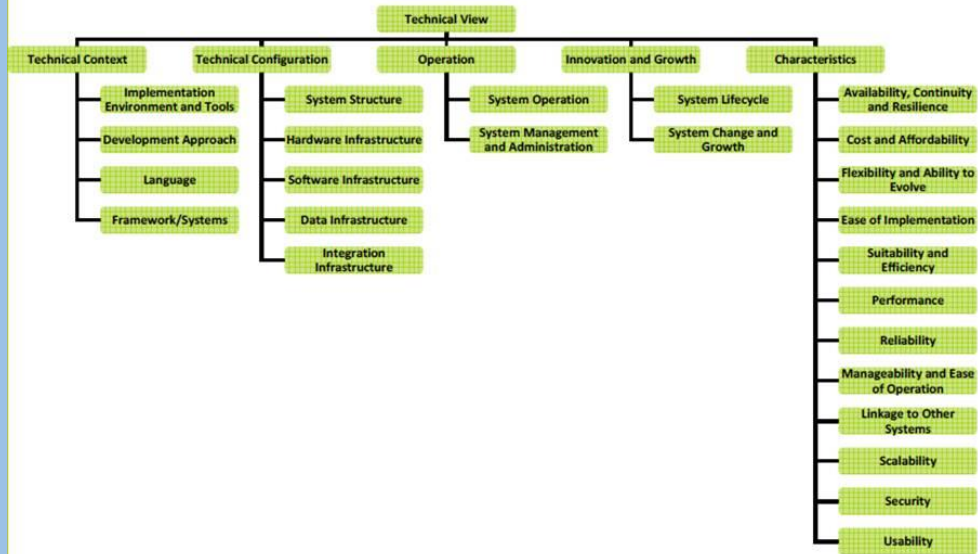
Infrastructure Configuration

Operations and Administration

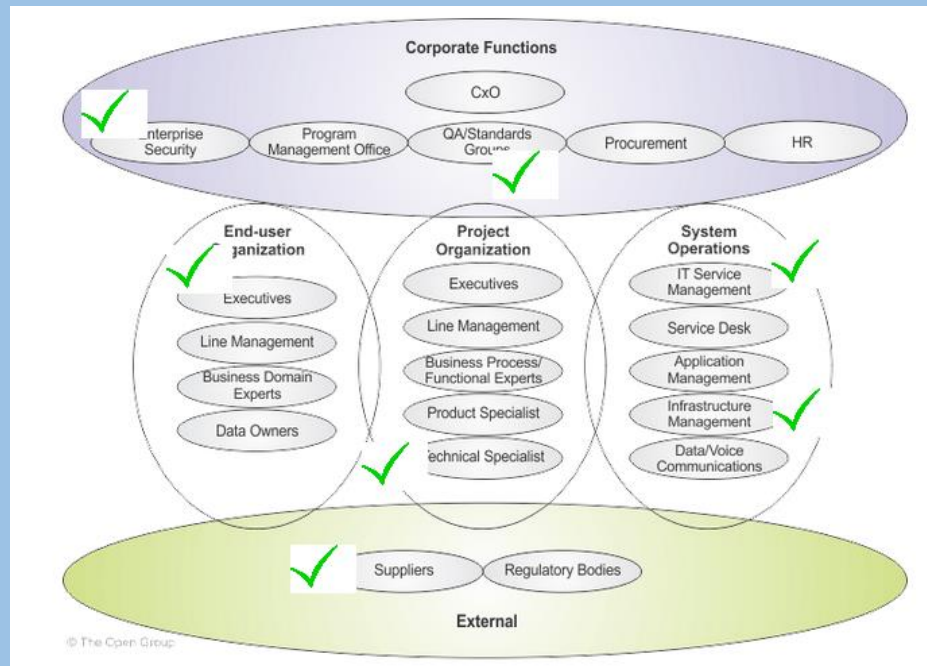
Lifecycle, Change and Growth

NFRs as a mainstay

Technical View Topics



Possible Stakeholders for Technology - Infrastructure Architecture :



Reference Architecture provides a generic way to move towards a solution

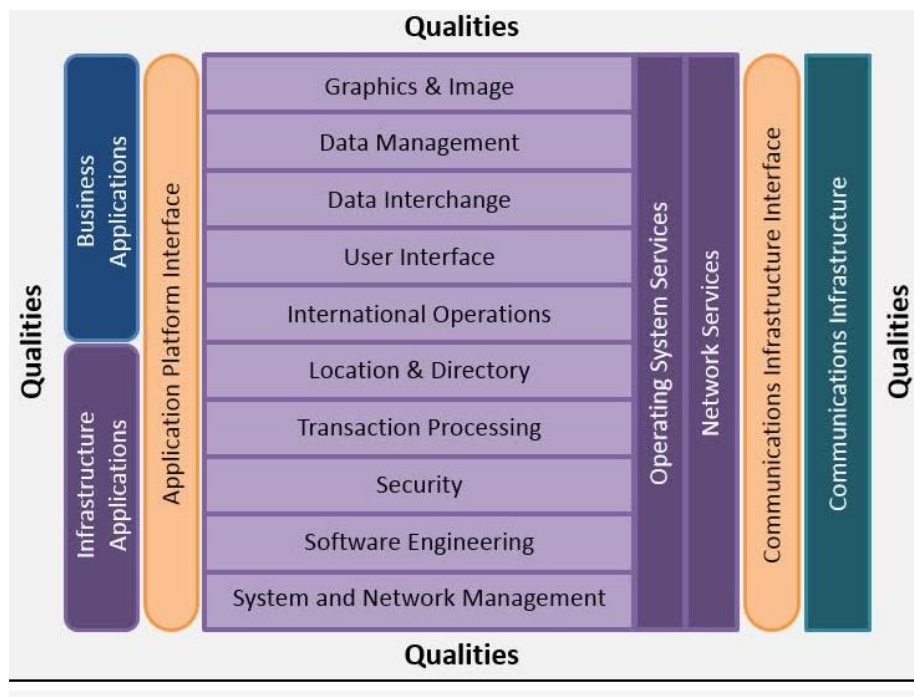
On the other hand, **Reference models**, referred often in ADM Phases of B, C and D, provide a more specific way to look at the solution space

TA Reference Models will include Generic, Foundation and ones that work on group[s] of Systems

Can extend to hosting styles including Cloud, Hybrid and on-premise.
Can be on migration to any of these as also on subscriptions based hosting including –As-A-service alternatives

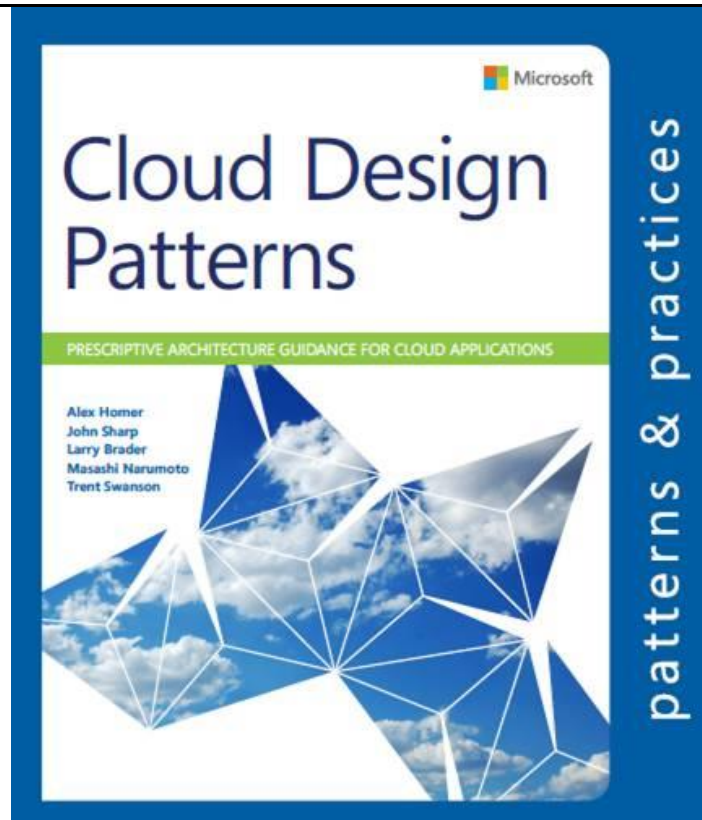
Select (TA) Reference Models :

- The TOGAF Technical Reference Model ([TRM](#))



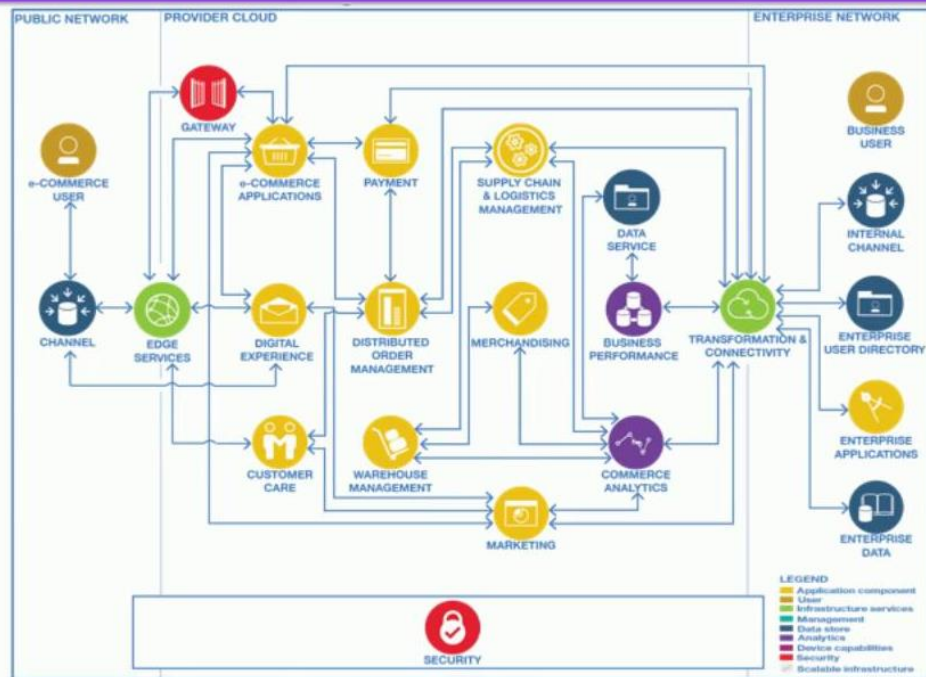
- [Generic technology models](#) relevant to the organization's industry "vertical" sector; for example, in the telecommunications industry such models have been developed by the TeleManagement Forum (TMF)
- [Technology models](#) relevant to Common Systems Architectures; for example, the III-RM – More relates to Application Architecture, but is also connected with Technology Architecture

Another Category of emerging Patterns, in tune with emerging trends :



Some of these Patterns apply to non-cloud situations also, and deal with Security and Identity, Tiers and Scalability and so on.

Cloud Customer Reference Architecture for e-Commerce



Develop Baseline Architecture : A Step in Technology Architecture

Baseline provides a starting point for the current task

TA gets baseline work pieces from Vision Phase, previous version slices of same Phase as also from work done so far in other Phases including Application Architecture as also from any previous study, papers and documents

Note that any Enterprise will have Infrastructure resources already installed and lying with additional capacity. The baseline materials should be able to give a clear picture of the same

Existing Technology Architecture Landscape

If an enterprise has existing technology and architecture descriptions, they should be used as the basis for the Baseline Description.

TA is given 0.1 of B D A in the vision as a Baseline to start with. Any existing study / documentation is also pointed out

For each version, Baseline can be :

Existing TA and any previous BA, AA,DA of completed versions

Existing studies, whitepapers and so on

For Portfolio 1 and Portfolio 2, all completed Business Architecture and Application / Data Architecture Artifact Building Blocks are the main source of Baseline Architecture.

Develop Target Architecture : A Step in Technology Architecture

Target View pieces : Building Blocks and supporting documentation for TA : Enterprise Architect can select from TOGAF suggested list, or from own experience and department practices

Target Architecture

: Wish To Be Technology Architecture Landscape

The Architectural details which will be prepared in the version installments during various visits to the TA Phase of this Architectural Initiative, for the portfolio of projects.

The EA has mandated preparation of Deployment Diagrams

Optionally Network Diagrams, Cloud Interface plans and Locations based Infra plans are suggested

The same, as given in different versions of DA work, as it iterates through ADM are :

Version 0.1 : Available in Phase A

Version 0.2 : ...

Version 0.3 : ...

...

...

Version 1.0 : Completed Building Blocks should depict entire eCommerce Technology (O/S and other Platform layer software, Infra and all hardware landscape) for this Portfolio as envisioned in Phase A documentation. These are ...

Perform Gap Analysis

– What is to be done in this Phase in this version slice ? What More later ?

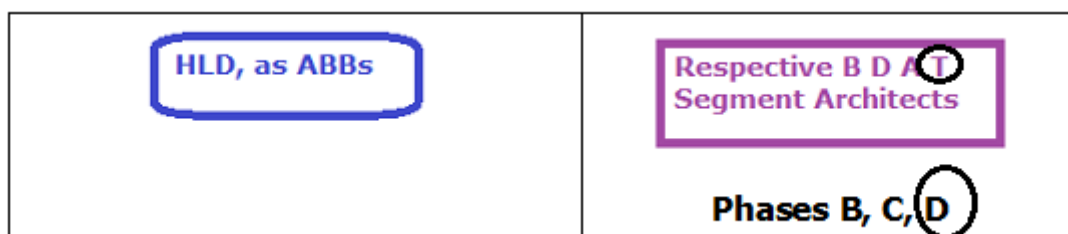
An important (and maximum time consuming) step in Technology Architecture

Gap Analysis and its performance (preparing the Building Blocks) is a natural progression of what has been achieved so far in BA, AA and DA provided Building Blocks

In TA work, it can be one appearing from a comparison of Baseline details and the shortfall in Requirement understating from the Stakeholder Engagement process, including those mentioned in the Vision Phase BBs and Business and Application Architecture / Data Architecture BBs

Performed Gap Analysis

Who Performed : The Technology Segment Architect



Is it accepted by Sr EA immediately ? **No**. Possibly, the work portion of this version slice will be accepted after the Stakeholder Review, which is a step appearing a little below.

Gap between BaseLine ---- > Target

A record of deviations from the planned architectural approach

for this Business Architecture Phase,
in this iteration

The scope of Infrastructure architecture, as part of the overall Enterprise Architecture, can be seen for the project that implements this Case Study are :

Ø Physical hardware and network : Needs hosting systems, connectivity hardware and so on at various points. Needs Centralized servers and Cloud solutions in areas such as :

Needs other automation devices including POS Terminals, Payment Acquisition systems and many more



Ø Security :

application itself would need security cover from virus and especially from Cross Scripting attacks when Web Browser is involved.

The application server / Cloud infrastructure will have to provide support by way of certificate stores, realms etc., for the various security measures discussed here. So the Portfolios require a Threat modeling exercise based on which rest of the issues discussed here on security is to be finalized.

Authentication : needs log-in mechanism such as password (with strong password mechanism), bio-metrics and certificate based systems. We need to distinguish the scenarios where humans get into the system and others where another software is establishing the handshake and choose one among the many authentication alternatives available.

Authorization : The study of such Use Cases will also reveal the role played by various human / external systems with respect to role based authorization.

Confidentiality and Message Integrity : These will come in when consideration of encrypting the transport channel or also the message itself are to be taken.

Ø **Storage** : The data storage, safe retrieval, and remote storage of data for retrieval in case of disasters like flood or earthquake etc, need suitable consideration. The SAN or NAS as discussed above will all need to be debated by the Architect in each Portfolio area.



Ø

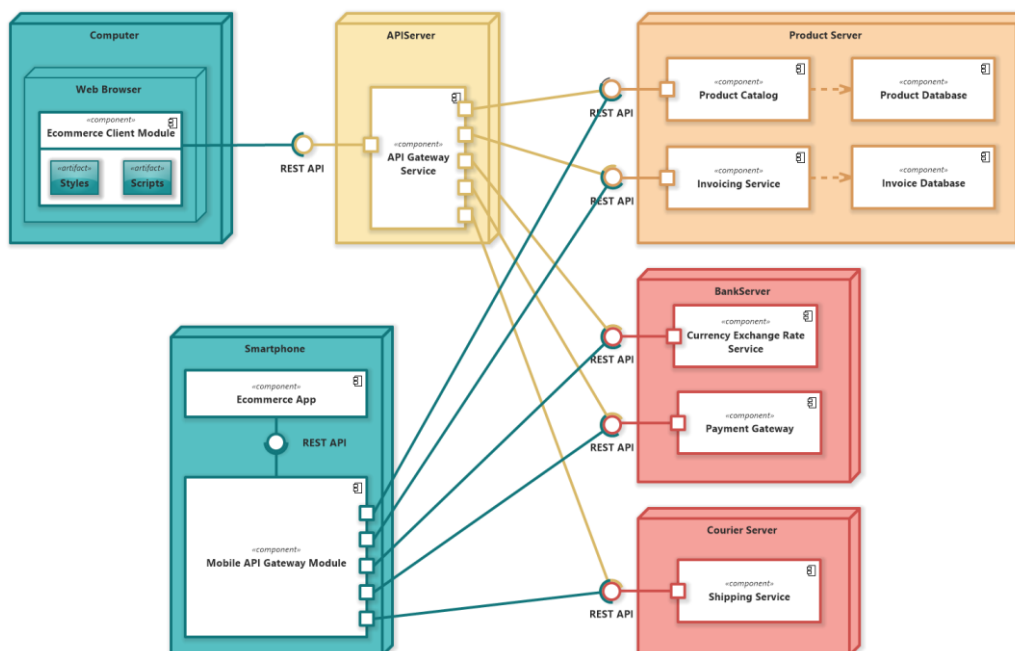
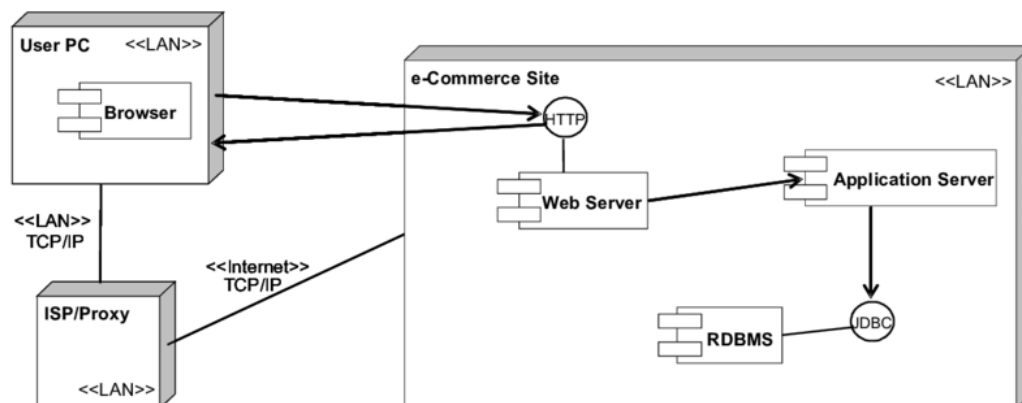
operating systems,
application platforms,
desktop operating systems,
messaging and enterprise services,
management,
or
operations.

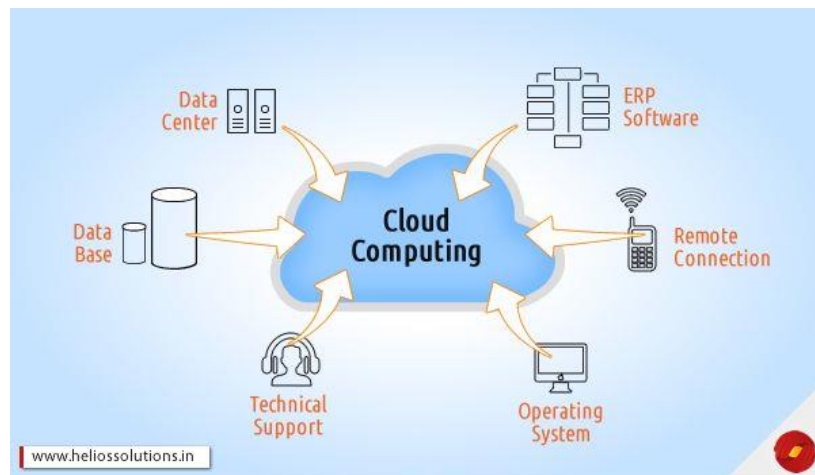
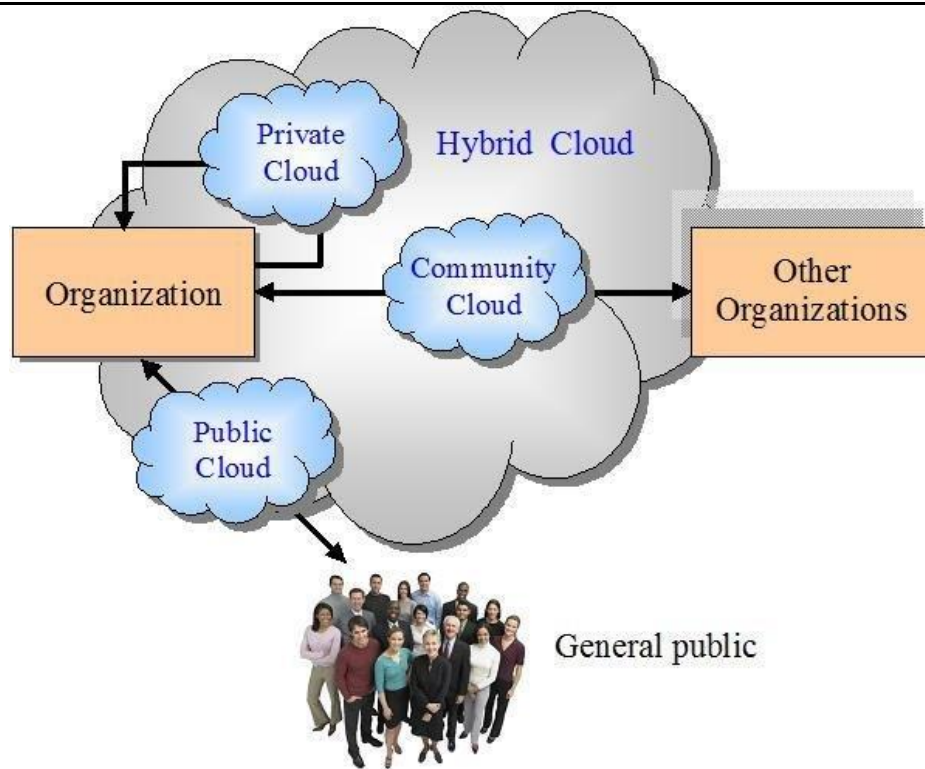
Some TA documentation based on AA :

Identify all Mid-Tier hosted Layers and all other tiers, based on AA :
Environment & Location

Platform Decomposition, : Identify Hosting software for each Layer in the
Mid-Tier

Communications Infrastructure : Identify appropriate Protocol
(supported by the target App Framework) for some inter-Tier and inter-
Layer connections





Roadmap is a **plan for business** or technology **change**, typically operating **across multiple disciplines over multiple years**.

Here a **Roadmap** is required to prioritize activities over the coming phases. It is the basis for a consolidated, cross-discipline roadmap that will evolve in Phase E.

Every version slice of Technology Architecture (0.2, 0.3 etc.,) **can update the progress** over such a Roadmap till all ABBs reach a plateau of 1.0 in all four segments.

Thereafter Phase E will take them forward to SBBs

Here a **Roadmap** is required to prioritize activities over the coming phases. It is the basis for a consolidated, cross-discipline roadmap that will evolve in Phase E.

Every version slice of Technology Architecture (0.2, 0.3 etc.,) **can update the progress** over such a Roadmap till all ABBs reach a plateau of 1.0 in all four segments.

Thereafter Phase E will take them forward to SBBs

Every version slice of Technology Architecture (0.2, 0.3 etc.,) can update the progress over such a Roadmap till all ABBs reach a plateau of 1.0 in all four segments.

Thereafter Phase E will take them forward to SBBs

Thereafter Phase E will take them forward to SBBs

Step : (In B, D A, T) : Here in Business Architecture :

Defined Business Roadmap

Which project, what **priority – relatively** for **portfolio of project initiatives**, **planned in this iteration**

Technology

Define SLA

People

Increase Staff

Security

Agree on

to increase the maturity level of the IT environment

to demonstrate the value of the modernization of the IT infrastructure to the Business.

Roadmap Components

Defined Business Roadmap

Which project, what priority – relatively for portfolio of project initiatives, planned in this iteration

Technology

Define SLA

People

Increase Staff

Security

Agree on

to increase the maturity level of the IT environment

to demonstrate the value of the modernization of the IT infrastructure to the Business.

Roadmap Components

Which project, what priority – relatively for portfolio of project initiatives, planned in this iteration

to demonstrate the value of the modernization of the IT infrastructure to the Business.

Roadmap Components

Technology

Define SLA

AI Using

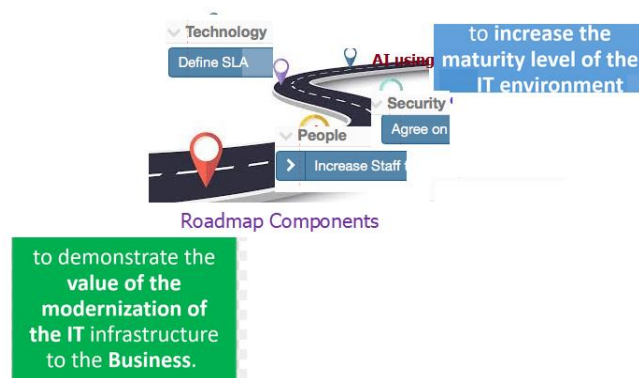
to increase the maturity level of the IT environment

Security

Agree on

People

Increase Staff



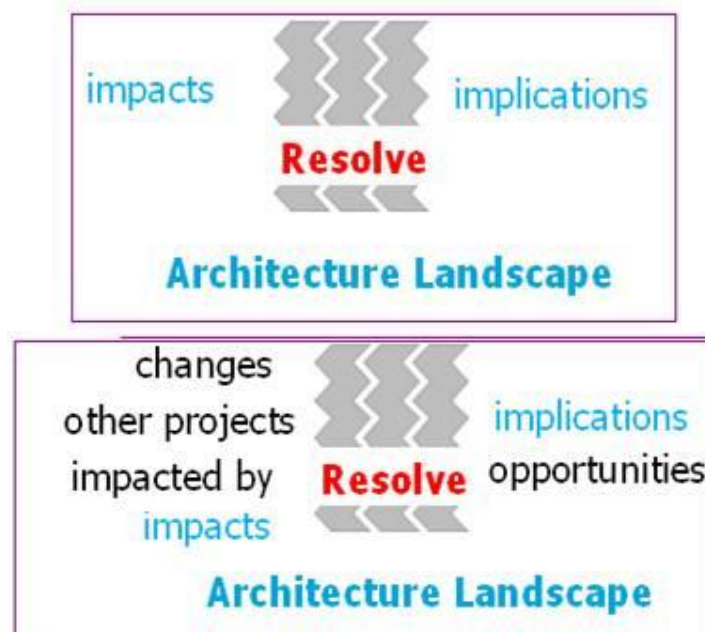
Impact is to be assessed and resolved across the four Domains of Architecture and corresponding Segment Architects.

TA will surely explain the Building Blocks prepared to BA, AA and DA and get their ideas of how this proposed Architectural work pieces will have impact on their work. Simultaneously the impact of Architectural work already done by them on current proposal is assessed.

The 'Resolution' happens by way of reworking the aspects of Technology Architecture which needs modification to minimize or avoid the impact.

Non-Functional Requirements, such as Security, Scalability and many others on Maintainability, Observability do expose impact areas that need special attention.

Resolved Impacts : Across Architecture Landscape. From BA angle :



Your TA discussed all diagrams with B, D and A Architects and arrives at a few impacts, including NFRs

A **Review Session with Stakeholders**, preferably at the end of every version slice work, is an **extension of the Stakeholder Engagement** that started in Vision Phase. A look at the Stakeholder Power – Interest Matrix will reveal the **list of such a Stakeholders to be contacted** at this stage

The Technology Segment Architect will explain the work done to the relevant Stakeholder and try to arrive at a **conscious understanding of mutual benefit**. It may need **more discussion to highlight** the impact of **Cloud Native, Digital client journey** and such other Infrastructure issues

Conducted Stakeholder Review of BT initiatives

Review : Original Stakeholder motivation for the architecture project initiatives

and the Statement of Architecture Work

prepared for proposed Business Architecture

The Technology Segment Architect Architect took the relevant details back to the LOB stakeholders and special Technology stakeholders including Security specialists and reviewed with them. This happens in every version of BA such as 0.2, 03. etc., till it reaches 1.0

Finalized Technology Architecture

Last step in Technology Architecture



It started in version 0.1 to take note of all existing Technology - Infrastructure related assets including platforms of Software, Hardware and Communication, as also Security and other supporting systems.

Very version slice added more Architectural details.

The Finalization as it happened in very version slice moved the Architectural artifacts of ABBs more closer to the Targets envisioned in the Portfolios.

Documents Produced in this Phase :

Architecture Definition Document : Plus its companion : Architecture Requirements Specification

This document cycles between Phases B, C and D and gets updates as the process gets rolling in that Phase over one or more iterations of this **Architectural Initiative**

Note : Building Blocks are the real architectural deliverables in these phases. The summary of the result is updates in this Architectural Definition Document

Building Blocks are placed in the Architectural Landscape portion of the Architecture Repository



Documents first produced Phase A, but gets updated for Technology Architecture portion in this Phase are:

Architecture Definition Document

Plus its companion :

Architecture Requirements Specification

More of Enterprise wide nature

- Technology standards catalog
- Technology portfolio catalog

More of Project Specific nature

- Application/Technology matrix

- Environments and Locations diagram
- Platform Decomposition diagram
- Processing diagram
- Networked Computing/Hardware diagram
- Network and Communications diagram

TOGAF Recommended Artifacts

Catalog as Building Blocks

Technology Standards Catalog

TOGAF Documentation says :

The Technology Standards catalog documents the agreed standards for technology across the enterprise covering technologies, and versions, the technology lifecycles, and the refresh cycles for the technology.

Depending upon the organization, this may also include location or business domain-specific standards information.

This catalog provides a snapshot of the enterprise standard technologies that are or can be deployed, and also helps identify the discrepancies across the enterprise.

The Technology Standards catalog contains the following metamodel entities:

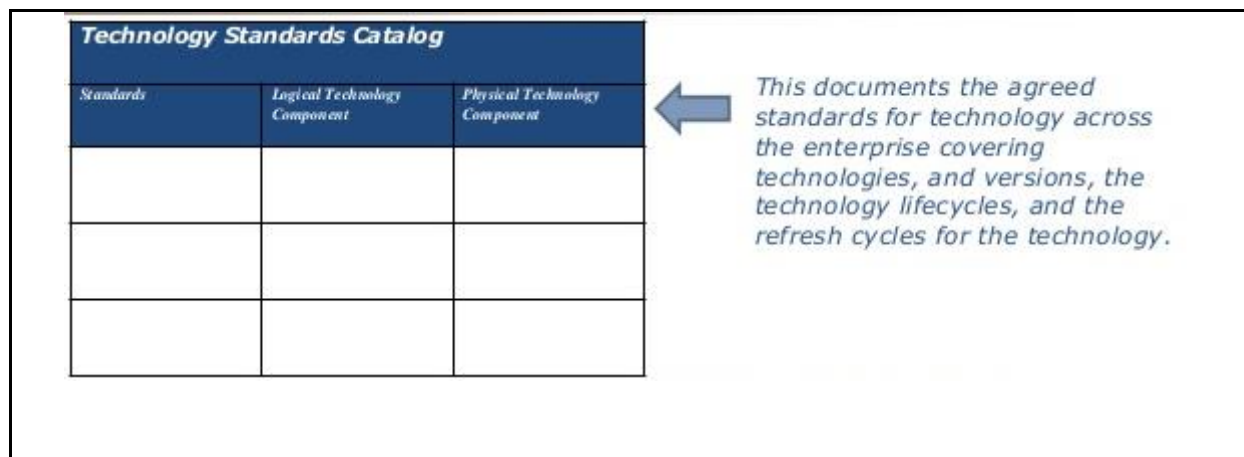
- Technology Service
- Logical Technology Component
- Physical Technology Component

If technology standards are currently in place, apply these to the Technology Portfolio catalog to gain a baseline view of compliance with technology standards.

Catalog	Purpose
Technology Standards Catalog	<p>This documents the agreed standards for technology across the enterprise covering technologies, and versions, the technology lifecycles, and the refresh cycles for the technology.</p> <p>It contains the following metamodel entities:</p> <ul style="list-style-type: none">•Platform Service, Logical Technology Component, Physical Technology Component

Standards	Logical Technology Component	Physical Technology Component
Code Development for core systems	Java using OpenJDK	JDK 16, with version revision considered with every release cadence plan
Web and complex Component Code Development	Jakarta EE on Glassfish Platform and Eclipse	Jakarta EE 9 Version revision considered only once a year
Development of Specific Performance Oriented Modules	GOLang using Visual Studio Code	Go release 1.13 with version revision considered with major releases only
RDBMS – SQL standard based	SQL Database with advanced modules	Oracle 19 c with version revision considered only with Long term Releases
No SQL	Document databases, Key-value databases, Wide-column stores, and Graph databases as needed	MongoDB, Casandra, Elasticsearch
Operating Environment	Windows	Windows server on premise and Azure
Operating Environment	Linux	Red Hat

This is only a sample list. One may like to add much more based on the experience and need, including those for hardware, network and so on. The idea is that Technology diversity is confined to the list only.



Technology Portfolio Catalog

TOGAF Documentation says :

The purpose of this catalog is to identify and maintain a list of all the technology in use across the enterprise, including hardware, infrastructure software, and application software. An agreed technology portfolio supports lifecycle management of technology products and versions and also forms the basis for definition of technology standards.

The Technology Portfolio catalog provides a foundation on which to base the remaining matrices and diagrams. It is typically the start point of the Technology Architecture phase.

Technology registries and repositories also provide input into this catalog from a baseline and target perspective.

Technologies in the catalog should be classified against the defined taxonomy in use in the enterprise, such as the TOGAF TRM - see the TOGAF® Series Guide: The TOGAF® Technical Reference Model (TRM) - adapted as necessary to fit the classification of technology products in use. These guides are not part of TOGAF 9.2 Certification but many such guides exist as part of enhanced learning for TOGAF Practitioners,

The Technology Portfolio catalog contains the following metamodel entities:

- Technology Service
- Logical Technology Component
- Physical Technology Component

Catalog	Purpose
Technology Portfolio Catalog	<p>The purpose of this catalog is to identify and maintain a list of all the technology in use across the enterprise, including hardware, infrastructure software, and application software. An agreed technology portfolio supports lifecycle management of technology products and versions and also forms the basis for definition of technology standards</p> <p>It contains the following metamodel entities:</p> <ul style="list-style-type: none"> •Platform Service, Logical Technology Component, Physical Technology Component

Platform Service	Provided by Logical Technology Component	Provided by Physical Technology Component
Coding for regular executable Services	Java , GoLang	JDK 16, Jakarta EE 9
Coding for Analysis Services	Jakarta EE on Glassfish Platform and Eclipse	Jakarta EE 9 Version revision considered only once a year
Databases, DWH infrastructure	Multi-purpose Database System	Oracle 19 c Client/ Server Architecture Large database and Space Management, Data Visualization Framework
RDBMS – SQL standard based	SQL Database with advanced modules	Oracle 19 c with version revision considered only with Long term Releases
No SQL	Non Relational data, Document databases,	MongoDB, Casandra,

	Key-value databases, Wide-column stores, and Graph databases as needed	Elasticsearch Each one can be leveraged suitably
Operating Environment	Windows	Windows server on premise and Azure subscription for Services
Operating Environment	Linux	Red Hat Enterprise Linux 8

This is only a sample list. One may like to add much more based on the experience and need, including those for hardware, network and so on. The idea is that Technology usage as per current Portfolios are confined to the list only, and any Technology Architecture should think beyond the list only in exceptional situations after due dispensation by the Architecture Governance board

The purpose of **Technology Portfolio catalog** is to identify and maintain a list of all the technology in use across the enterprise, including hardware, infrastructure software, and application software.



Technology Portfolio Catalog		
	[provided by]	[realized in]
Platform Service	Logical Technology Component	Physical Technology Component

Matrices as Building Blocks

Application / Technology Matrix

TOGAF Documentation says :

The Application/Technology matrix documents the mapping of applications to technology platform.

This matrix should be aligned with and complement one or more platform decomposition diagrams.

The Application/Technology matrix shows:

- Logical/Physical Application Components
- Services, Logical Technology Components, and Physical Technology Components
- Physical Technology Component realizes Physical Application Component relationships

This is also known as System / Technology Matrix

System/Technology Matrix

- The System/Technology matrix documents the mapping of business systems to technology platform.
- The System/Technology matrix shows:
 - Logical/Physical Application Components
 - Services, Logical Technology Components, and Physical Technology Components
 - Physical Technology Component *realizes* Physical Application Component relationships

Slide ©2009-2011 The Open Group. All Rights Reserved
114

TOGAF®

Example System/Technology Matrix

LOGICAL APPLICATION COMPONENT	PHYSICAL TECHNOLOGY COMPONENT	SERVER ADDRESS	IP ADDRESS
ABM	Web server - node 1	F01ws001@host.com	10.xx.xx.xx
	Web server - node 2	F01ws002@host.com	10.xx.xx.xx
	Web server - node 3	F01ws003@host.com	10.xx.xx.xx
	App server - node 1	F02as001@host.com	10.xx.xx.xx
	App server - node 2	F02as002@host.com	10.xx.xx.xx
	App server - node 3	F02as003@host.com	10.xx.xx.xx
	Database server (production)	F02dbp001@host.com	10.xx.xx.xx
	Database server (staging)	F03dbs001@host.com	10.xx.xx.xx
Load balancer and Dispatcher	Dispatcher server	F03nd001@host.com	242.xx.xx.xx

Slide ©2009-2011 The Open Group. All Rights Reserved
115

TOGAF®

Example System/Technology Matrix

TECH FUNCTION	HARDWARE LOGICAL	HARDWARE PHYSICAL	SOFTWARE LOGICAL	SOFTWARE PHYSICAL
Load balancing	<ul style="list-style-type: none"> Name – Balancer Vendor - IBM Server Type – eServer Clustered – No No. of Nodes – N/A Server logical address - d04lb01@host.com Maintenance Window – Sun 0100 to 0300 	<ul style="list-style-type: none"> Model/Type – IBM P7xx Serial Number – 1S4568 Processor Type - RISC Power p5 Number of Processors - 4 way Memory - 8GB Hard drive - 4 TB IP - 11.xx.xx.xx 	<ul style="list-style-type: none"> Product- IBM Load balance manager Vendor - IBM OS – UNIX based 	<ul style="list-style-type: none"> SW Components – LB v3.2 (list all the other components of the SW product) AIX 10.2.1 License Type - Enterprise wide license License expiry date - 12/31/2011

Example System/Technology Matrix

APPLICATION COMPONENT	DEPLOYMENT UNIT	TECHNOLOGY COMPONENT
•Load Balancer	•Smart dispatch v1.2 (both installation and execution code)	•Load balancing server (d03lb001@host.com)
•Commerce pages	•HTML code •Applets •JSP	•Web Server cluster (d03ws001@host.com, d03ws002@host.com, d03ws003@host.com)
•Commerce Engine	•Order Entry (both installation and execution code) •Shopping Cart (both installation and execution code)	•Application Server (d03as001@host.com, d03as002@host.com)

Slide ©2009-2011 The Open Group. All Rights Reserved
117

TOGAF®

Application Component	Technology Details : IP address etc.,	Deployment Details As part of which executable . identifiable unit
Order – App Component	Order Control executable enabled using ... platform and ... server / cloud hardware
Order Placement App Component	Order Control executable enabled using ... platform and ... server / cloud hardware
UICustomer	Hosted in desktop as ... And in Mobile and handheld devices as .. And on wearable devices as ...
Fulfilment Component	''' '''	Physical actual control by Fulfillment automation units

System Technology Matrix			
Logical Application Component	Physical Technology Component	Server Address	IP Address
ABM	Webserver node 1	R01ws001@host.com	10.xx.xx.xx
	Webserver node 2	R01ws002@host.com	10.xx.xx.xx
	Webserver node 3	R01ws003@host.com	10.xx.xx.xx
	App server node 1	R02as001@host.com	10.xx.xx.xx
	App server node 2	R02as002@host.com	10.xx.xx.xx
	App server node 3	R02as003@host.com	10.xx.xx.xx
	Database (prod)	R03dbp001@host.com	10.xx.xx.xx
	Database (staging)	R03dbst001@host.com	10.xx.xx.xx
Load balancer and dispatcher	Dispatcher Server	R03md001@host.com	242.xx.xx.xx
...			


The System/Technology matrix documents the mapping of business systems to technology platform.

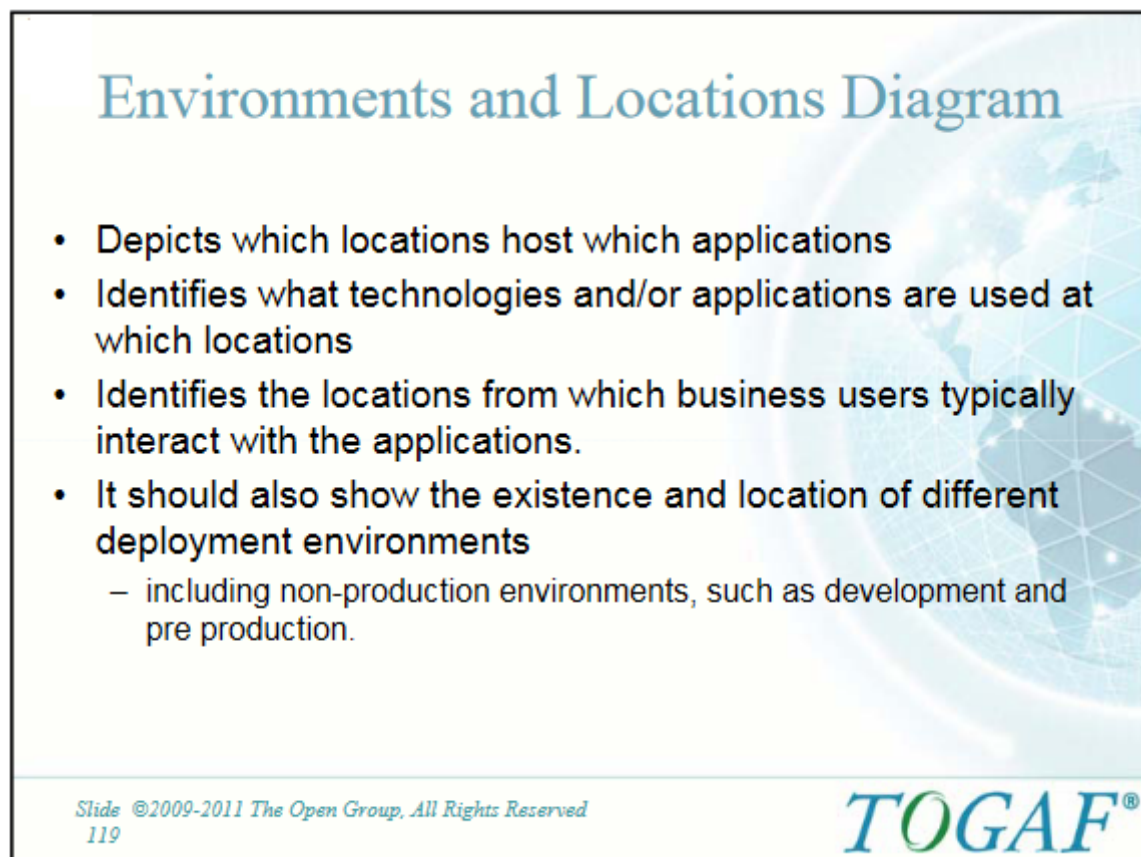
Diagrams as Building Blocks

Environments and Locations Diagram

TOGAF Documentation says :

The Environments and Locations diagram depicts which locations host which applications, identifies what technologies and/or applications are used at which locations, and finally identifies the locations from which business users typically interact with the applications.

This diagram should also show the existence and location of different deployment environments, including non-production environments, such as development and pre-production.

The slide features a light blue background with a faint world map and network lines. The title 'Environments and Locations Diagram' is at the top in a large, blue, serif font. Below it is a bulleted list of five points. The first four points are in a bold, black, sans-serif font, and the fifth point is followed by a sub-point in a regular, black, sans-serif font. At the bottom left, there is a small line of text in a blue, italicized, sans-serif font. At the bottom right, the 'TOGAF' logo is displayed in a large, blue, serif font with a registered trademark symbol.

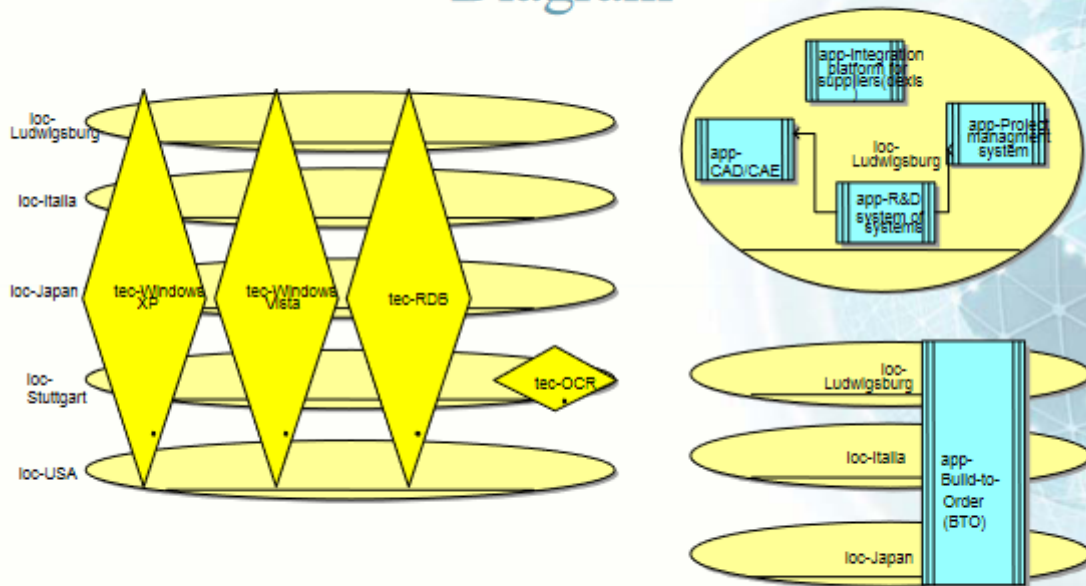
Environments and Locations Diagram

- Depicts which locations host which applications
- Identifies what technologies and/or applications are used at which locations
- Identifies the locations from which business users typically interact with the applications.
- It should also show the existence and location of different deployment environments
 - including non-production environments, such as development and pre production.

Slide ©2009-2011 The Open Group, All Rights Reserved
119

TOGAF®

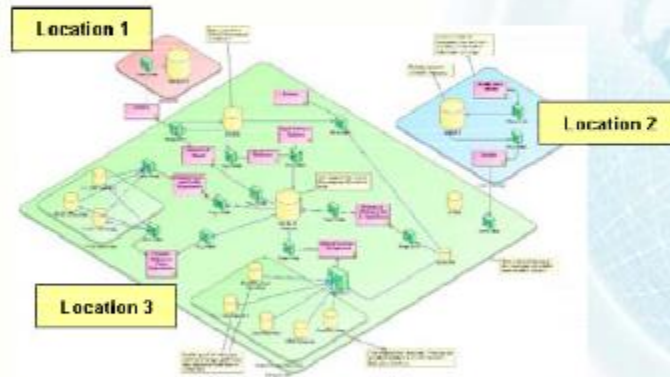
Example Environments and Locations Diagram



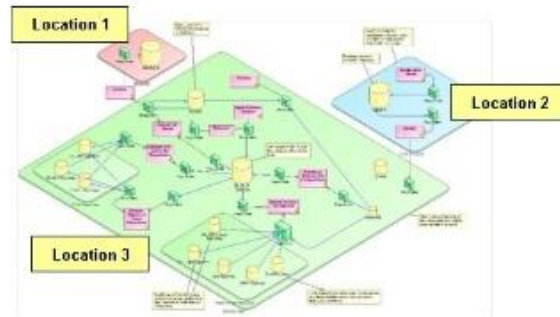
Slide ©2009-2011 The Open Group, All Rights Reserved
120

TOGAF®

Environments and Location Diagram



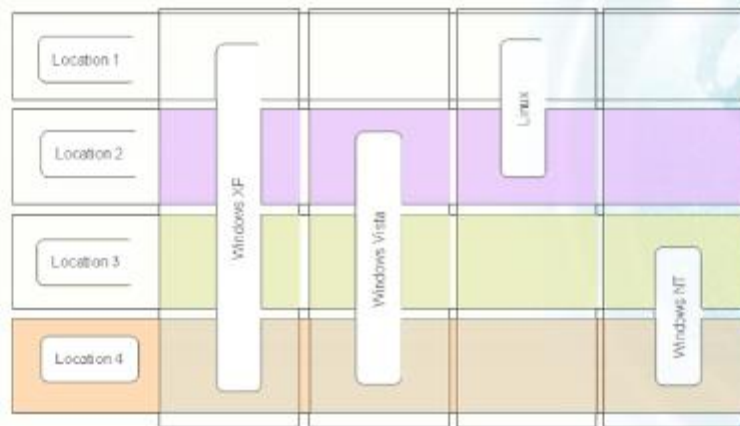
Environments and Location Diagram



Environments and Location Diagram



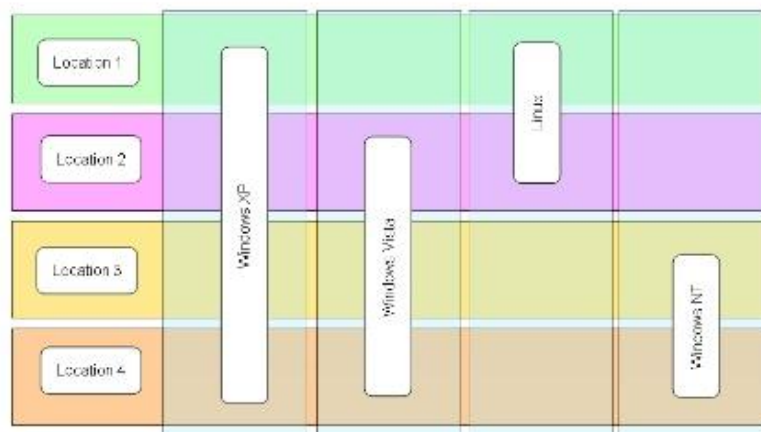
Environments and Location Diagram

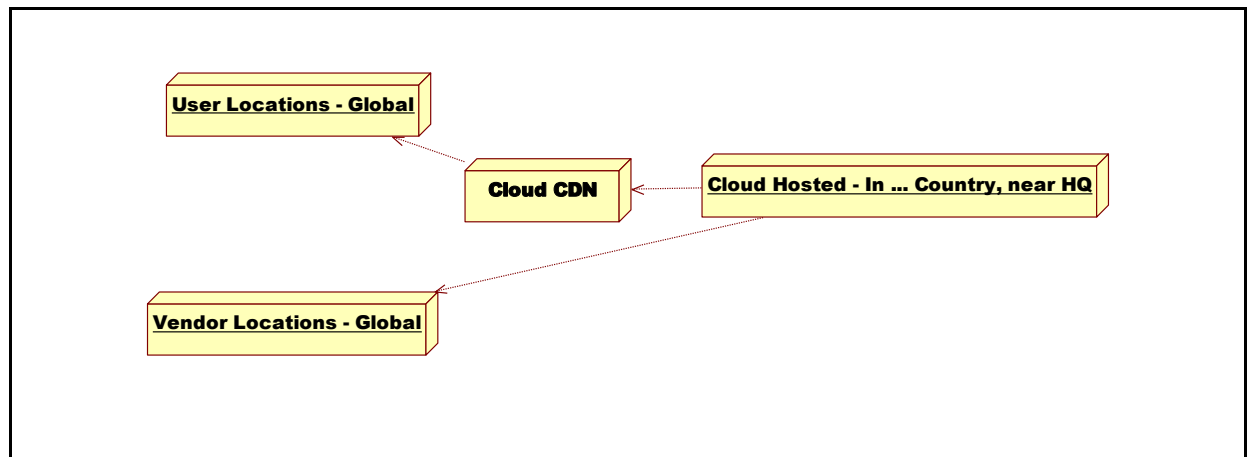


Slide ©2009-2011 The Open Group, All Rights Reserved
123

TOGAF®

Environments and Location Diagram





Platform Decomposition Diagram

TOGAF Documentation says :

The Platform Decomposition diagram depicts the technology platform that supports the operations of the Information Systems Architecture. The diagram covers all aspects of the infrastructure platform and provides an overview of the enterprise's technology platform. The diagram can be expanded to map the technology platform to appropriate application components within a specific functional or process area. This diagram may show details of specification, such as product versions, number of CPUs, etc. or simply could be an informal "eye-chart" providing an overview of the technical environment.

The diagram should clearly show the enterprise applications and the technology platform for each application area can further be decomposed as follows :

- Hardware:
 - Logical Technology Components (with attributes)
 - Physical Technology Components (with attributes)

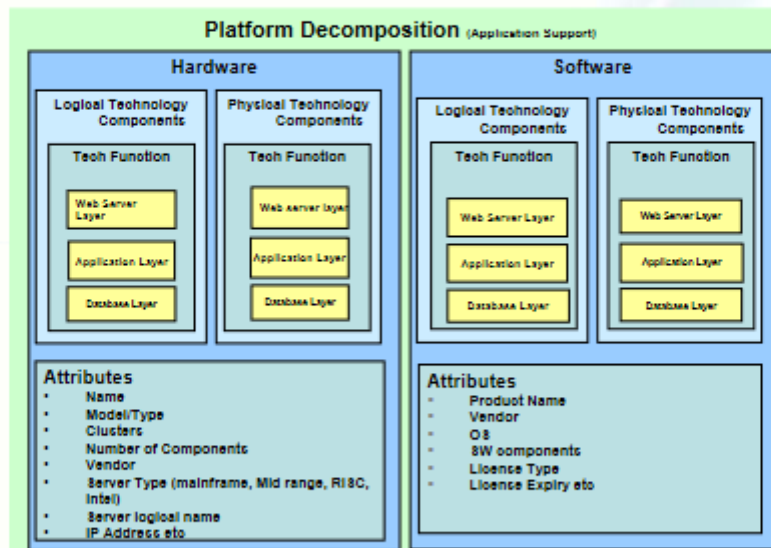
- Software:
 - Logical Technology Components (with attributes)
 - Physical Technology Components (with attributes)

Depending upon the scope of the Enterprise Architecture work, additional technology cross-platform information (e.g., communications, telco, and video information) may be addressed.

Platform Decomposition Diagram

- The Platform Decomposition diagram depicts the technology platform that supports the operations of the Information Systems Architecture.
- The diagram covers all aspects of the infrastructure platform and provides an overview of the enterprise's technology platform.

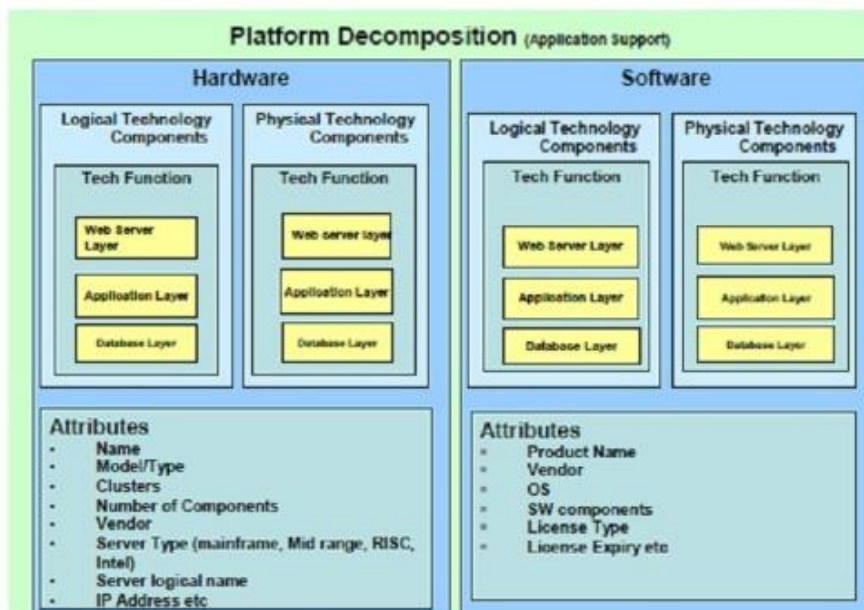
Example Platform Decomposition Diagram



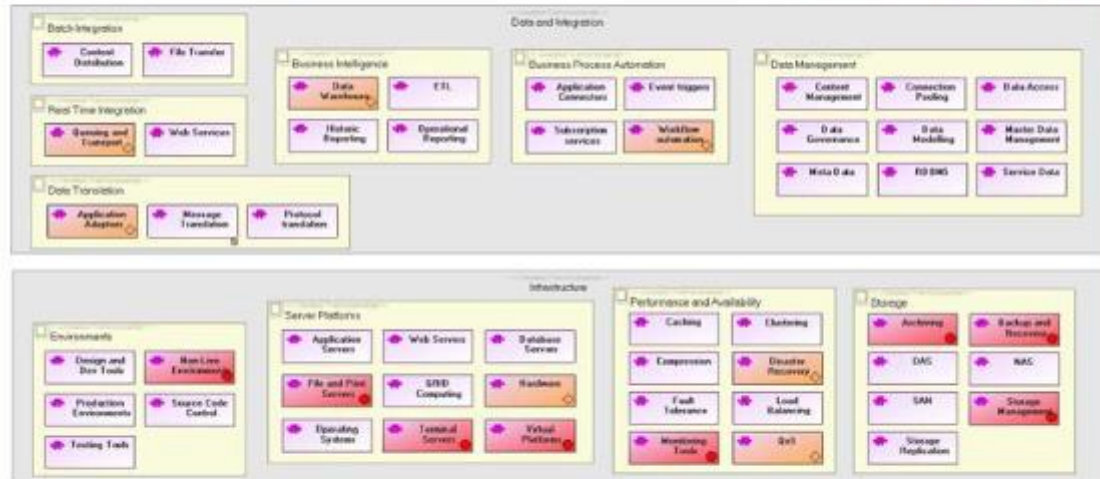
Slide ©2009-2011 The Open Group. All Rights Reserved
125

TOGAF®

Platform Decomposition Diagram



Platform Decomposition Diagram



Platform Decomposition Diagram

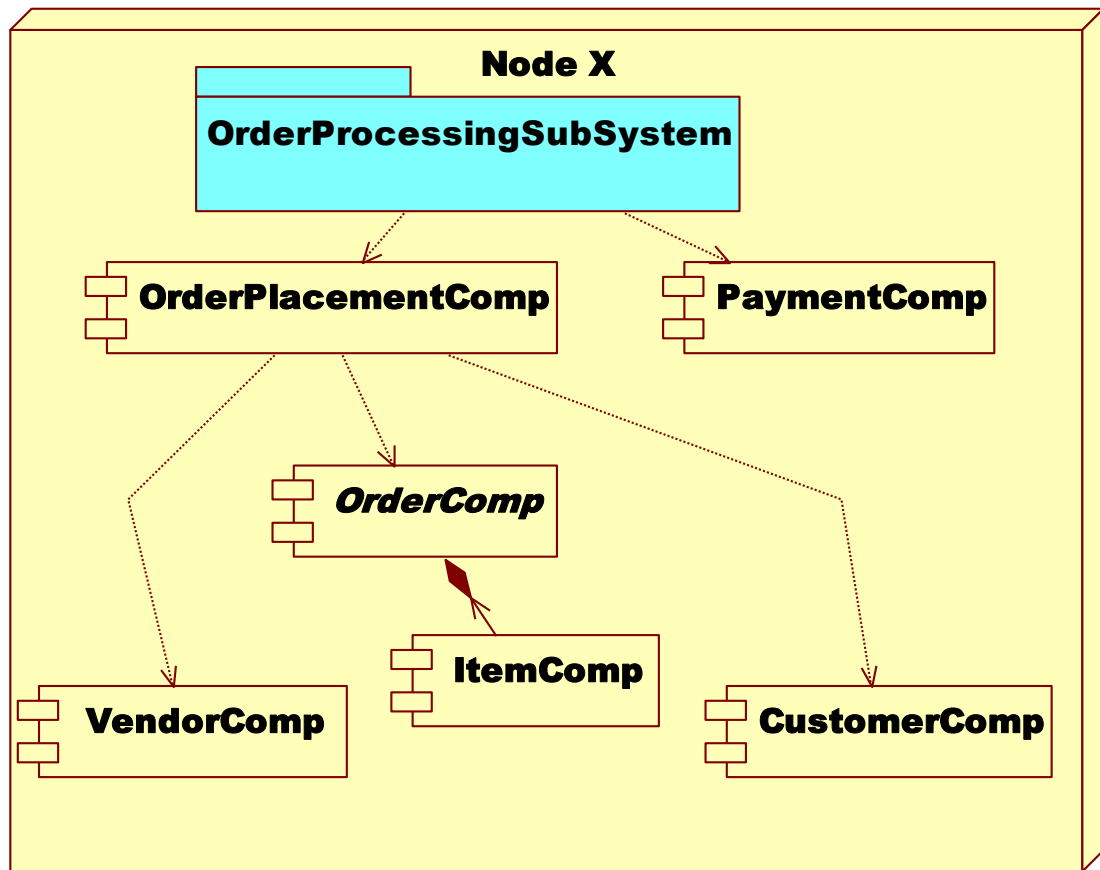


Slide ©2009-2011 The Open Group. All Rights Reserved
127

TOGAF®

Platform Decomposition Diagram





Processing Diagram

TOGAF Documentation says : The Processing diagram focuses on deployable units of code/configuration and how these are deployed onto the technology platform. A deployment unit represents grouping of business function, service, or application components. The Processing diagram addresses the following:

- Which set of application components need to be grouped to form a deployment unit
- How one deployment unit connects/interacts with another (LAN, WAN, and the applicable protocols)
- How application configuration and usage patterns generate load or capacity requirements for different technology components

The organization and grouping of deployment units depends on separation concerns of the presentation, business logic, and data store layers and service-level requirements of the components. For example, the presentation layer deployment unit is grouped based on the following:

- Application components that provide UI or user access functions
- Application components that are differentiated by location and user roles

There are several considerations to determine how application components are grouped together. Each deployment unit is made up of sub-units, such as:

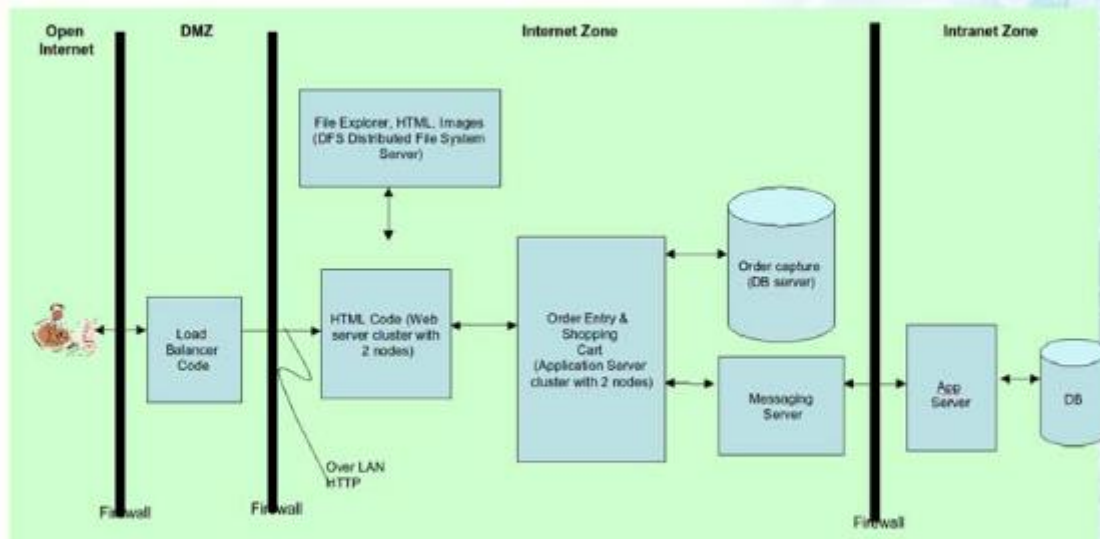
- Installation: part that holds the executable code or package configuration (in case of packages)
- Execution: application component with its associated state at run time
- Persistence: data that represents the persistent state of the application component

Finally, these deployment units are deployed on either dedicated or shared technology components (workstation, web server, application server, or database server, etc.). It is important to note that technology processing can influence and have implications on the services definition and granularity.

Processing Diagram

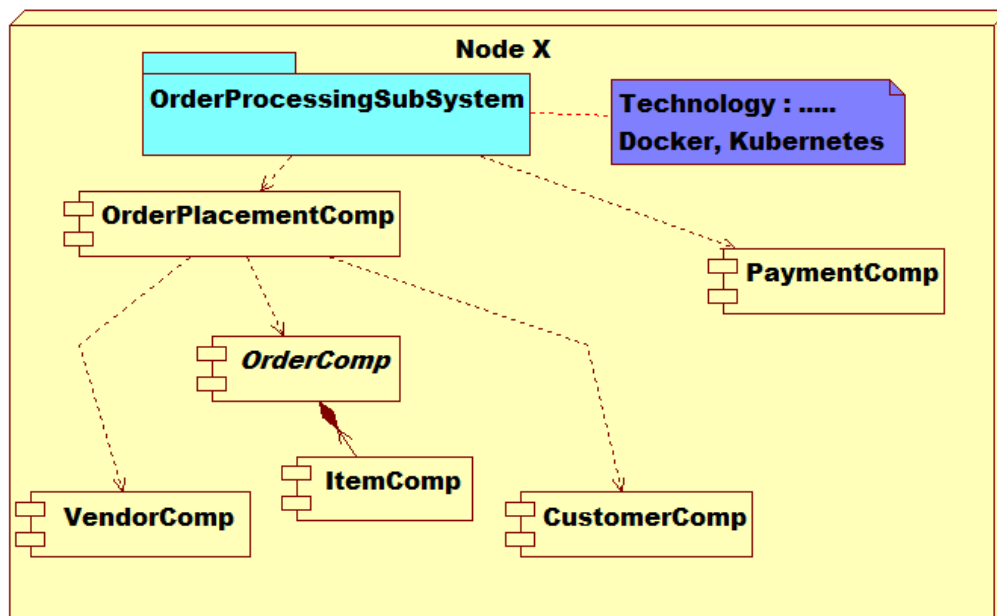
- The Processing diagram focuses on deployable units of code/configuration and how these are deployed onto the technology platform.
- The Processing diagram addresses the following:
 - Which set of application components need to be grouped to form a deployment unit
 - How one deployment unit connects/interacts with another (LAN, WAN, and the applicable protocols)
 - How application configuration and usage patterns generate load or capacity requirements for different technology components
- The organization and grouping of deployment units depends on separation concerns of the presentation, business logic, and data store layers and service-level requirements of the components.

Example Processing Diagram



Slide ©2009-2011 The Open Group. All Rights Reserved
129

TOGAF®



Networked Computing / Hardware Diagram

TOGAF Documentation says :

Starting with the transformation to client-server systems from mainframes and later with the advent of e-Business and J2EE, large enterprises moved predominantly into a highly network-based distributed network computing environment with firewalls and demilitarized zones. Currently, most of the applications have a web front-end and, looking at the deployment architecture of these applications, it is very common to find three distinct layers in the network landscape; namely a web presentation layer, a business logic or application layer, and a back-end data store layer. It is a common practice for applications to be deployed and hosted in a shared and common infrastructure environment.

So it becomes highly critical to document the mapping between logical applications and the technology components (e.g., server) that supports the application both in the development and production environments. The purpose of this diagram is to show the "as deployed" logical view of logical application components in a distributed network computing environment. The diagram is useful for the following reasons:

- Enable understanding of which application is deployed where in the distributed network computing environment
- Establishing authorization, security, and access to these technology components
- Understand the Technology Architecture that supports the applications during problem resolution and troubleshooting
- Isolate performance problems encountered by applications, determine whether it is application code-related or technology platform-related, and perform necessary upgrade to specific physical technology components
- Identify areas of optimization as and when newer technologies are available which will eventually reduce cost
- Enable application/technology auditing and prove compliance with enterprise technology standards
- Serve as an important tool to introduce changes to the Technology Architecture, thereby supporting effective change management

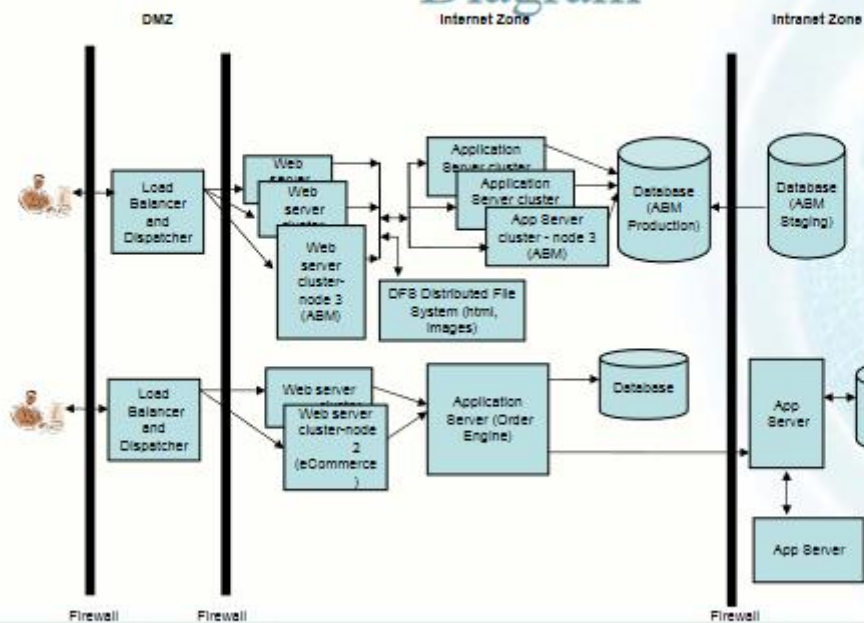
- Establish traceability and changing application end-point address while moving application either from a shared environment to a dedicated environment or vice versa

The scope of the diagram can be appropriately defined to cover a specific application, business function, or the entire enterprise. If chosen to be developed at the enterprise level, then the network computing landscape can be depicted in an application-agnostic way as well.

Network Computing Hardware Diagram

- The purpose of this diagram is to show the "as deployed" logical view of logical application components in a distributed network computing environment.
- The diagram is useful for the following reasons:
 - Enable understanding of which application is deployed where
 - Establishing authorization, security, and access to these technology components
 - Understand the Technology Architecture that support the applications during problem resolution and troubleshooting
 - Isolate performance problems encountered and perform necessary upgrade to specific physical technology components
 - Identify areas of optimization
 - Enable application/technology auditing and prove compliance
 - Serve as an important tool supporting effective change management

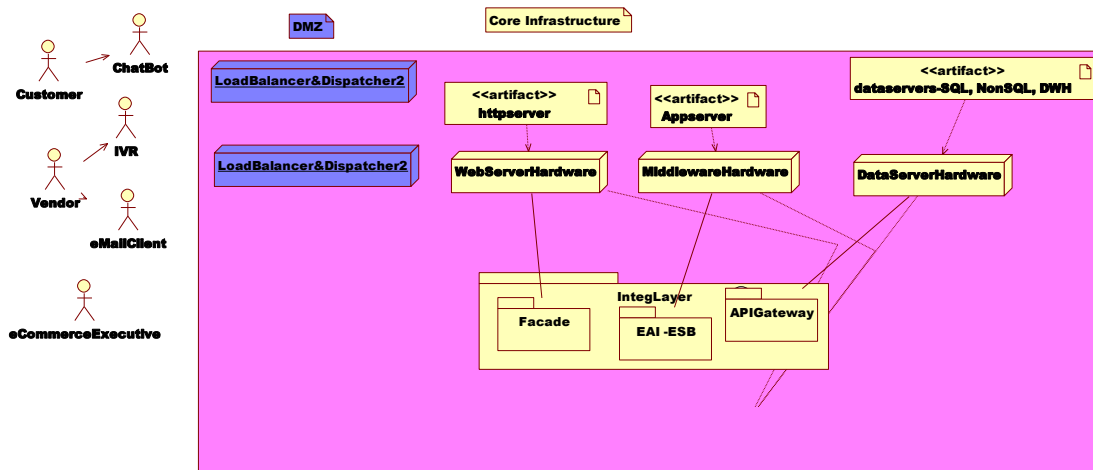
Example Network Computing Hardware Diagram



Slide ©2009-2011 The Open Group, All Rights Reserved
131

TOGAF®

See how more Actors, more of Logical Components are added here :



Network and Communications Diagram

TOGAF Documentation says :

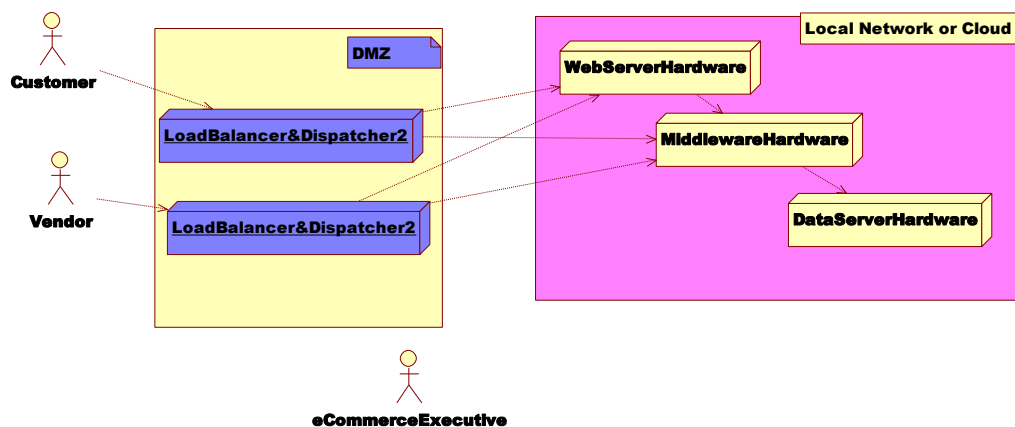
The Network and Communications diagram describes the means of communication - the method of sending and receiving information - between these assets in the Technology Architecture; insofar as the selection of package solutions in the preceding architectures put specific requirements on the communications between the applications.

The Network and Communications diagram will take logical connections between client and server components and identify network boundaries and network infrastructure required to physically implement those connections. It does not describe the information format or content, but will address protocol and capacity issues.

This is also known as Communications Engineering Diagram

Communications Engineering Diagram

- The Communications Engineering diagram describes the means of communication between assets in the Technology Architecture
- It takes logical connections between client and server components and identifies network boundaries and network infrastructure required to physically implement those connections.
- It does not describe the information format or content, but addresses protocol and capacity issues.

[illegible]

Following were the notes prepared by the Technology Architect and the Enterprise Architect at various stages :

Infrastructure Facilities

An organization of any appreciable size will typically have a number of facilities or locations that it operates. These will include structures such as: corporate offices, data centers, disaster recovery centers, shop fronts, warehouses and a variety of specific locations such as hotels, airports, transmission centers, vehicles, vessels, carparks and more.

These locations will typically need to be modeled at a business level to model concerns such as where business is performed, the location of processes and personnel and which locations are required for particular capabilities and business functions for example - what business processes are performed in an ambulance.

Similar models will be required at Data Architecture level to indicate where data is created, stored and utilized for example - in which locations can a patient record be created. At an Application Architecture level the relationship between applications and particular locations will be important for example - which applications are used in robot assisted warehouses and which applications are required for a pop-up kiosk.

Cloud Computing Service Model

The most important differentiation of the cloud is the service model. The service model defines how to offer cloud to the customer. Every model has different characteristics.

Service models include :

- **SaaS (software as a service):** The SaaS model allows users to design and promote their software. Based on the cloud's infrastructure, this means consumers essentially share some resource on the network and run their software on that resource. Consumers are not able to manage the underlying infrastructure, such as the network, operating system, etc., and they are able only to use the software. The software is accessible on different platforms, such as mobile phone, tablet, or computer. Good examples of SaaS include Google Docs, Office 365, and Zendesk, with which the consumer can access software from different platforms with, in most cases, only the requirement of an Internet connection.
- **PaaS (platform as a service):** The PaaS model offers the user the ability to deploy his/her own software, library, database, and everything else required for the software, in a cloud environment, without thinking about the underlying infrastructure. With PaaS, the consumer can develop an application, using the language provided, and essentially not care about the resource. An example of PaaS is Heroku, in which a consumer develops an application using his or her own preferred language, and the only work required is to create the software infrastructure, such as the database, and deploy the software for allowing the user to access his/her cloud resource.
- **IaaS (infrastructure as a service):** The IaaS model is the most manageable service model. With IaaS, a user can run a preferred software. The software is run in the infrastructure the user creates. This includes the operating system, applications, or anything else that is required. At the same time, the user is able to manage some underlying network features, such as, for example, the firewall and load balancer. Good examples of IaaS include GCP, AWS, and Azure.

The Deployment Models

Private: The private model of cloud computing is intended for internal use only. This kind of cloud is normally built to be shared in a private company, across the different business units. This cloud is completely owned and managed by the provider, normally, another business unit of the company.

- **Community:** This kind of cloud model is intended to be shared across a specific community, for example, a security community or charity. In this type of deployment, the community owns and manages the cloud infrastructure.
- **Public:** This cloud model can essentially be used by anyone. Usually, it is rented. Examples of this kind of model are Amazon Web Services, GCP, and Microsoft Azure.
- **Hybrid:** This type of cloud model combines different models. It is normally used when there is a need for part of the data to remain private, for example, customer data, for security reasons. The purpose of having a hybrid cloud model is to combine characteristics of different kinds of cloud models, for example, a need for privacy but, at the same time, a need to share some information with a community.

Multi-tenant

Despite the fact that you are aiming for greater team autonomy, it does not necessarily follow that there must be separate application-owned infrastructure components. There are still options so that you can best use the underlying resources. Technically, you may create a fresh infrastructure for the queues and topics that are needed by an application, but that makes sense only at a certain scale. You also must be able to provision these capabilities easily on a multi-tenant infrastructure, whether self-managed, cloud managed, or even on an appliance. Having a choice of consistency is critical to meet the different needs of the applications.

Multicloud

This multi-tenant aspect becomes more important in a multi cloud environment where asynchronous communication is required among many different cloud domains that are run by different cloud vendors. Being able to use capability on any cloud makes it easier to rapidly satisfy the needs of teams, projects, and solutions that are distributed across multiple cloud domains.

Using a cloud-based infrastructure, scaling can be horizontal by adding and removing servers or containers and adopting more cost-effective pricing models. You can now deploy thin slivers of application logic on minimalist run times into lightweight independent containers.

Furthermore, rather than running a pair of servers as with traditional HA, it is trivial to run many servers in containers to limit the effects of one container failing. By using container orchestration frameworks such as K8s, you can introduce or dispose of containers rapidly to scale up or down workload.

Introduction to Docker

Docker is probably the most famous software for containerization. It offers a level of virtualization called operating-system-level virtualization. This is known as containerization.

This type of isolation allows us to run more than one OS inside another. For example, it is possible to create a container inside an Ubuntu Linux with Red Hat. There is an important distinction between a container and a virtual machine (VM), which is that a container doesn't require a full operating system for run. When we create a VM, we essentially re-create an entire OS. When we create a container, we get only a part of the operating system. This reduces the size of the image. When we talk about virtualization, we can identify two types: hypervisor-based virtualization and operating-system virtualization. In hypervisor-based virtualization, the system emulates the hardware. This means that we can re-create the network, the hard disk drive (HDD), etc.

In operating system virtualization, the virtualization is made at the operating-system level. The host isolates each container from the other, in particular, the host isolates the filesystem of each container, but they run in a single host. Because the container has a filesystem isolated OS, a container loses flexibility.

You can run containers only with the same host. It is not possible to run Windows on a Linux host, for example, because the OS virtualization is run by the systems. Windows and Linux have two different OS kernels and filesystem structures, which does not allow Windows to run on Linux.

Containers suffer from less security compared to hypervisor virtualization. This is because when we create a Linux container, we use libcontainers. These access five basic namespaces—Network, Process, Mount, Hostname, and Shared Memory—but, for example, don't use SELinux, cgroups, and other libraries used to enhance OS security.

This translates to a greater possibility of malicious software preventing execution and operation. Conversely, a container is an isolated environment. This means that in the case of a breach on the container, it is not likely to create an issue on the host system, because a container does not share anything with the hosting OS.

Docker is composed of the following different components :

- The Docker engine is a client-server application. The client talks with the server application, called a daemon. The daemon is responsible for executing the container. Client and daemon can be on the same machine or different ones.
- Images are the basis of our Docker architecture. Images are used to launch our container, and we can create personal images, starting with another basic image.
- The registry is where the image is stored. We can identify two types of registries: private and public. Public registries can be reached via the address <https://hub.docker.com>. This is the official Docker Hub registry. Here, we can create an account and start to store our images in the registry.
- Containers are basically an image that is executed. A container can have more than one running process inside, depending on how the images are designed and created. A container is essentially a lightweight stand-alone application. We can combine more than one container to execute a complex application.

Kubernetes, sometimes referred to as K8s, is an open source platform developed by Google and managed by the Cloud Native Computing Foundation. It is used to manage and scale containerized applications in a cluster, such as Docker.

Kubernetes defines some basic building blocks for creating and managing a resource. All the components are designed to be loosely coupled. The first component of the Kubernetes building block is called a pod. A pod is a collection of Docker containers, all with the same IP address. Every pod can be composed of one or more containers, hosted in the same host and sharing the resource. Each pod shares a unique IP address with the cluster and can be managed manually, by the API or by the controller.

The other resource defined in Kubernetes is called labels. These are used for creating a key-pair value for identifying the resource, for example, a pod. The labels don't provide any uniqueness. In Kubernetes, it is possible to have more than one object with the same name and to create what is called a label selector, or selector.

With a selector, it is possible to identify a set of objects. The selector is the core primitive for grouping objects in Kubernetes. Actually, we can use two types of selectors, equality-based and set-based, when we want to identify a resource. A label selector can be made with multiple requirements, separated by commas. In this case, all requirements must be met.

The last piece of the building block is services. In Kubernetes, a service is a set of pods working together. For example, we can create a pod with a complete piece of our microservice architecture and define a service with that. The service is defined and identified by a label. This helps to identify the service itself. With Kubernetes, it is possible to search a service with the service discovery and use the IP address or the DNS name for finding the service we need to manage.

Kubernetes is a container orchestrator for automating deployment, management, and scaling of the containerized application, such as, for example, Docker. This container is defined in pods. This offers a high level of abstraction, grouping the containerized applications. A pod is the essential building block of every Kubernetes application. Pods help us to release our container.

Instead of releasing a single container at a time, we can release the pods. A pod can have more than one container in the definition, which helps in releasing more than one container at a time.

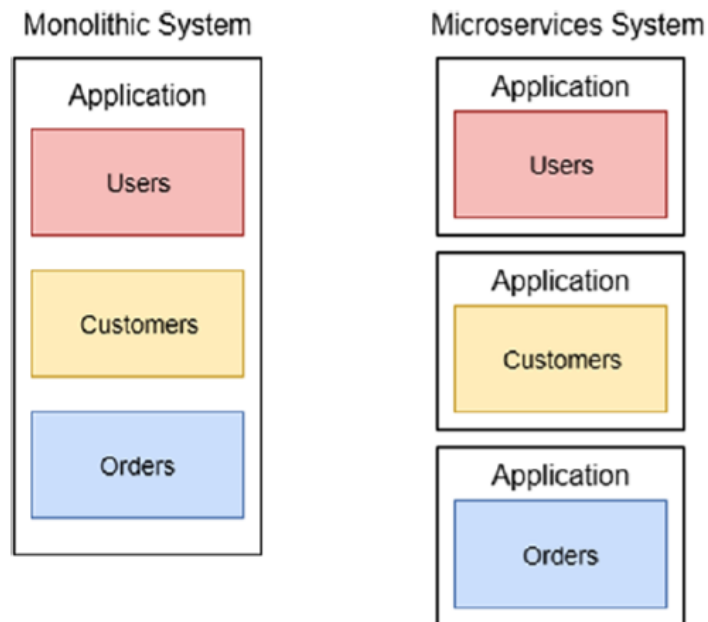
With Kubernetes, we can create a cluster to manage the system. A cluster is a collection node. A Kubernetes cluster has one master node and zero or more worker nodes. This means that we can have a Kubernetes cluster with just one master node. In Kubernetes, a node is a machine in which it is possible to run and schedule a pod. The machine can be physical or virtual. Because the node is essentially the basic block of the cluster, every node can be a master node or a worker node. It depends only on the configuration we choose for our cluster.

One of the key components of Kubernetes is etcd. This is a lightweight distributed and persistent key-store developed by CoreOS. This component is used to store the state of the cluster at any time. etcd is used to define a high-availability cluster. The reason for using Kubernetes can be found in the nature of DevOps itself. When a company decides to adopt the DevOps practice, with Kubernetes we can faster release and deploy our application using Docker

At the same time, Kubernetes can grow with business necessity. This is because of the scalability of the service itself. With Kubernetes we can add or remove a node to the cluster, and Kubernetes takes care to manage the pods with our application.

Microservices Architecture with Docker and Kubernetes

When we consider and design an architecture using Kubernetes, the natural choice is Docker. With Docker, we can create a level of isolation we need for our microservices architecture. The first thing to bear in mind in the design of the architecture we want to implement is how it is different from a monolithic architecture.

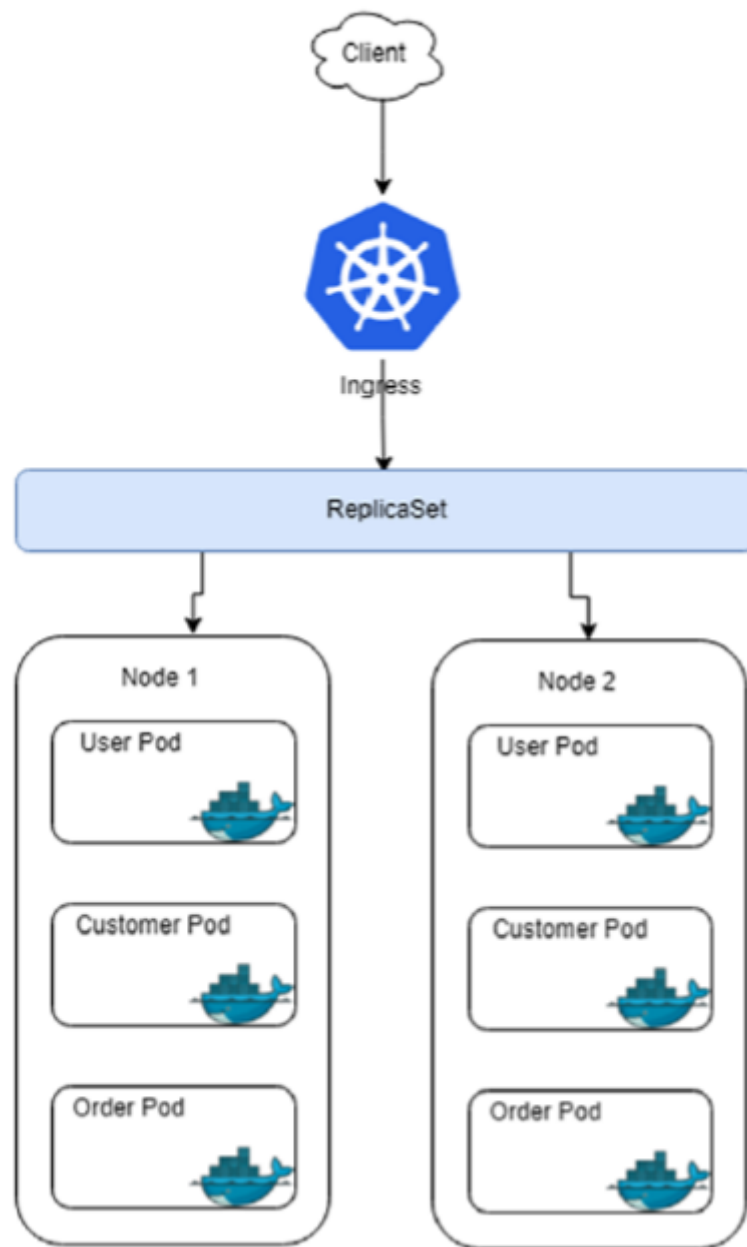


monolithic system vs. a microservices system

When we move from a monolithic application to a microservices application, we must think about each piece of the system as a separate application.

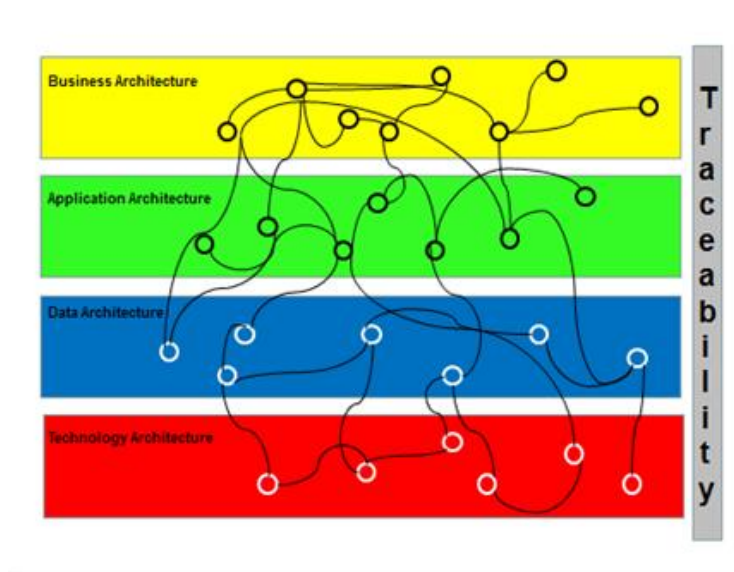
Each application, for example, for orders, customers, and users, is not a simpler component of the application but a small application itself. This is the first major change that we must reflect in our architecture. To achieve that shift, we can use Docker and Kubernetes.

This kind of application can be defined by Container as a Service (CaaS), in which each Docker is a single application, and Kubernetes helps to manage them all.



Kubernetes microservices architecture

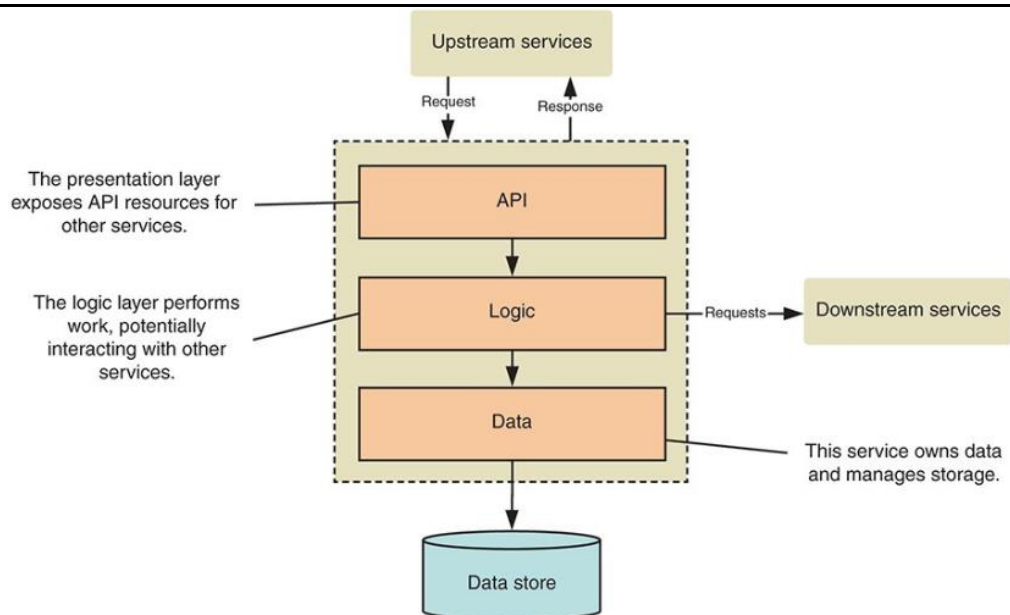
Impact on any pre-existing architectures ?



However, microservices are more than scalable components – they are software building blocks that can be developed, maintained, and deployed independently of each other. Splitting development and maintenance among modules that can be independently developed, maintained, and deployed improves the overall system's CI/CD cycle/

Since each microservice is deployed independently from the others, there will not be any binary compatibility or database structure compatibility constraints. Therefore, there is no need to align the versions of the different microservices that compose the system. This means that each of them can evolve, as needed, without being constrained by the others.

Assigning their development to completely separate smaller teams, thus simplifying job organization and reducing all the inevitable coordination inefficiencies that arise when handling large teams.



The high-level architecture of an individual microservice

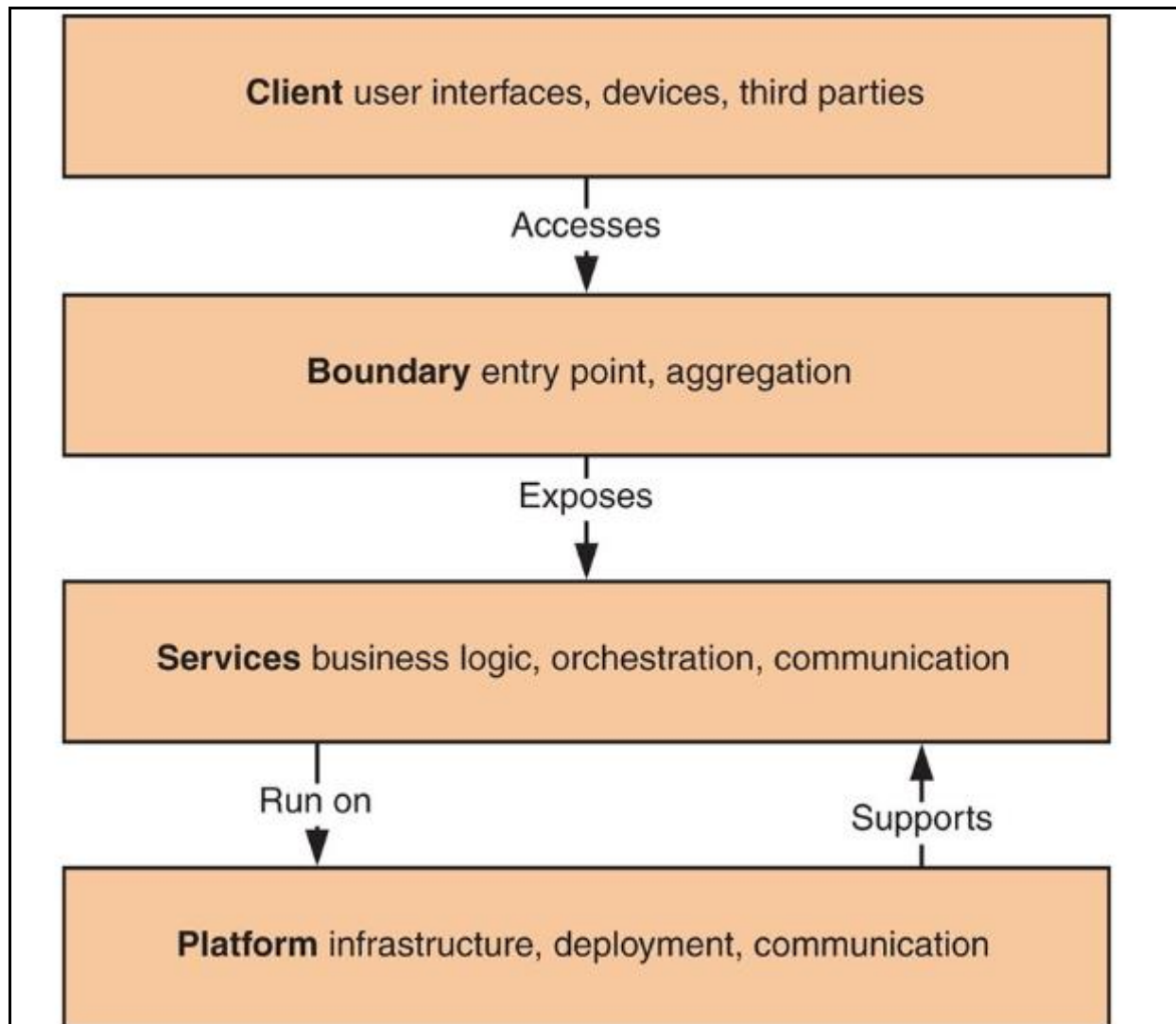
Four-tier model for a microservice application :

Platform—A microservice platform provides tooling, infrastructure, and high-level primitives to support the rapid development, operation, and deployment of microservices. A mature platform layer enables engineers to focus on building features, not plumbing.

Services—In this tier, the services that you build interact with each other to provide business and technical capabilities, supported by the underlying platform.

Boundary—Clients will interact with your application through a defined boundary that exposes underlying functionality to meet the needs of outside consumers.

Client—Client applications, such as websites and mobile applications, interact with your microservice backend.



Each layer is built on the capabilities of the layers below; for example, individual services take advantage of deployment pipelines, infrastructure, and communication mechanisms that the underlying microservice platform provides. A well-designed microservice application requires sophistication and investment at all layers.

A microservice Infrastructure platform : A microservice is supported by infrastructure.

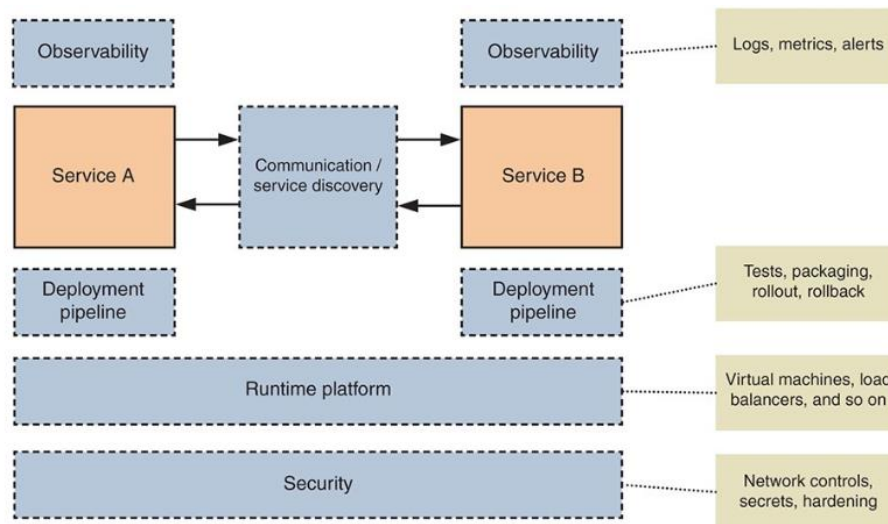
A deployment target where services are run, including infrastructure primitives, such as load balancers and virtual machines

Logging and monitoring aggregation to observe service operation

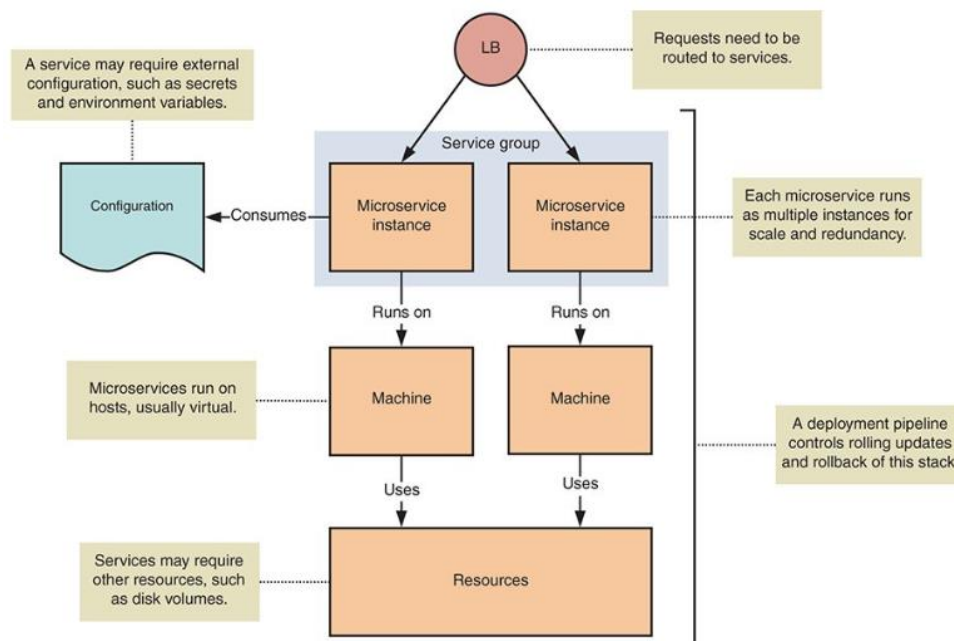
Consistent and repeatable deployment pipelines to test and release new services and versions

Support for secure operation, such as network controls, secret management, and application hardening

Communication channels and service discovery to support service interaction



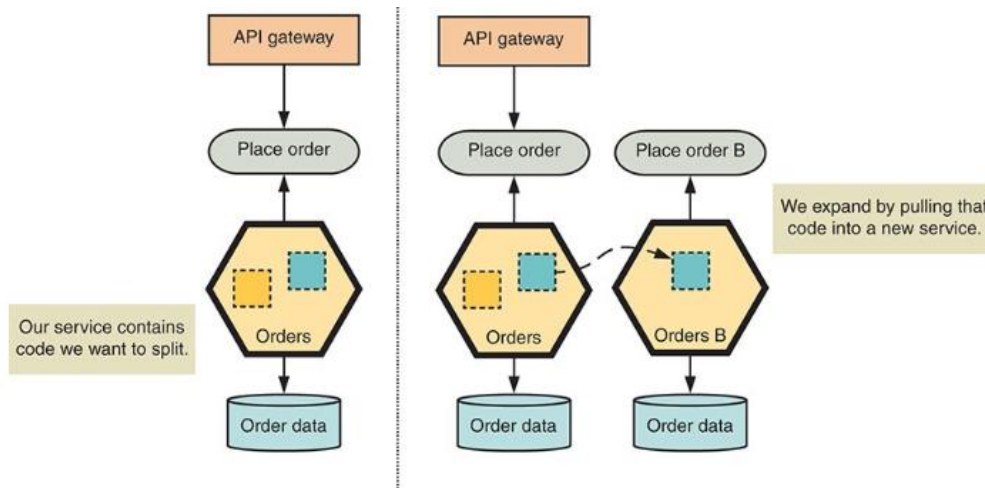
If each microservice is a house, then the platform provides roads, water, electricity, and telephone cables



A deployment configuration for a microservice running in a typical cloud environment

MIGRATION :

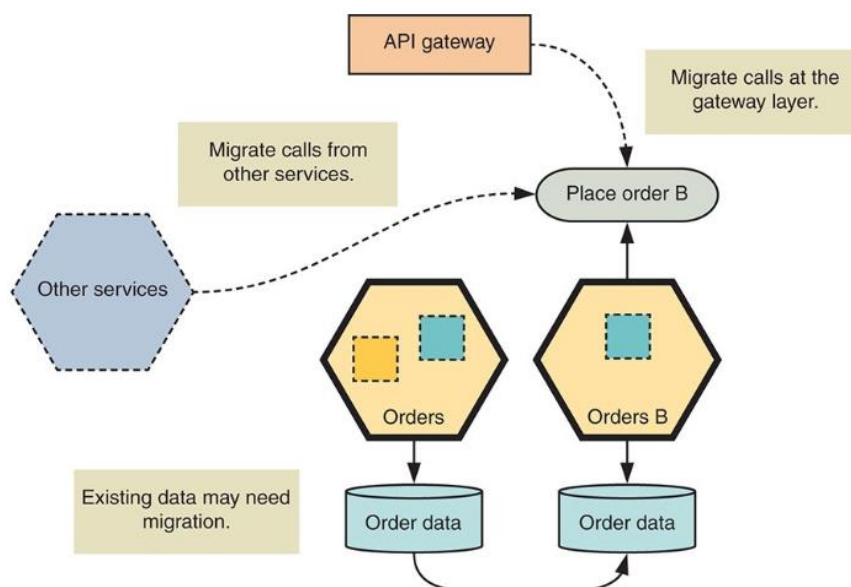
To carve out new services, you should apply the expand-migrate-contract pattern



Expanding functionality from one service into a new service

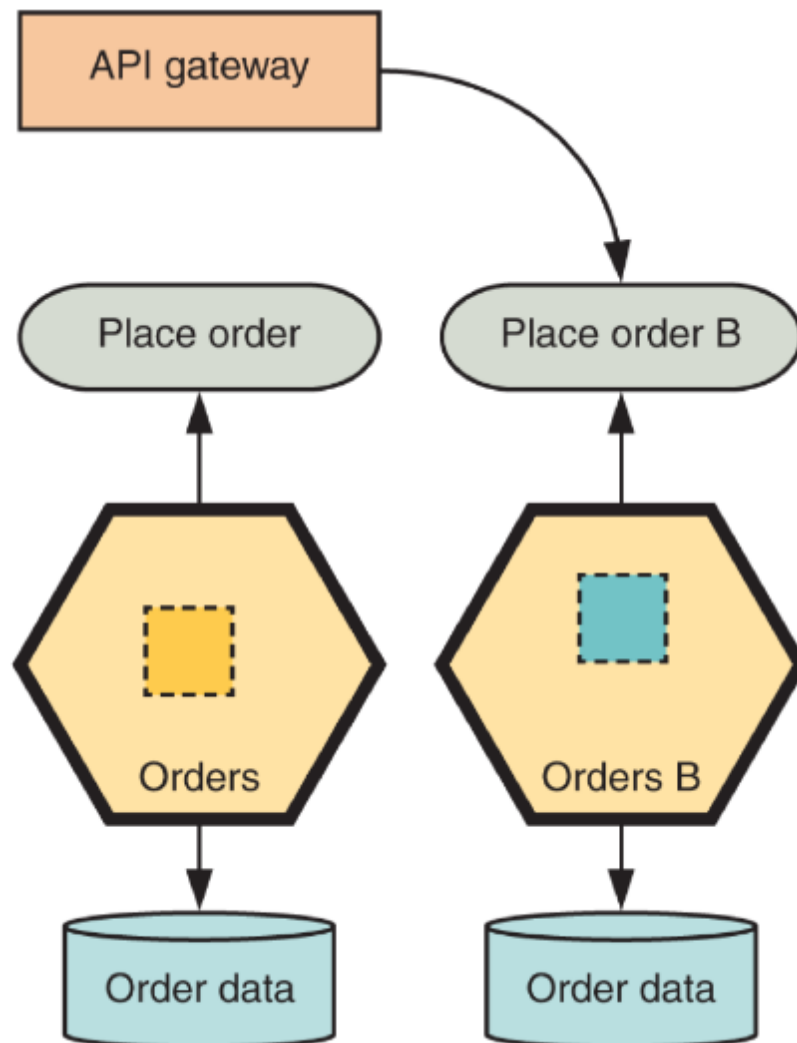
When you need to expand —pull the target functionality into a new service

Next, you need to migrate consumers of your old service to the new service. If access is through an API gateway, you can redirect appropriate requests to your new service :



Migrating existing consumers to the new service

To complete the process, you have one last step. Finally, you can contract the original service, removing the now obsolete code :



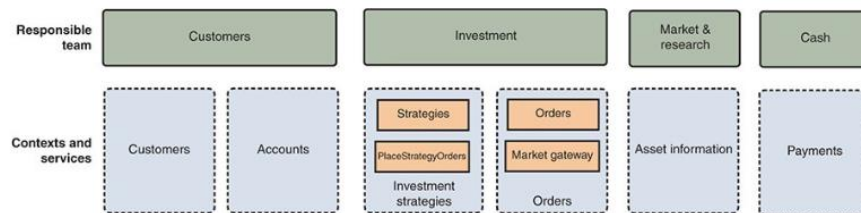
In the final state of your service migration, you've contracted your service to remove functionality that now resides in the new service

Service ownership in organizations :

In a large organization, different teams will own different microservices. This isn't a bad thing —it's an important part of scaling as an engineering team

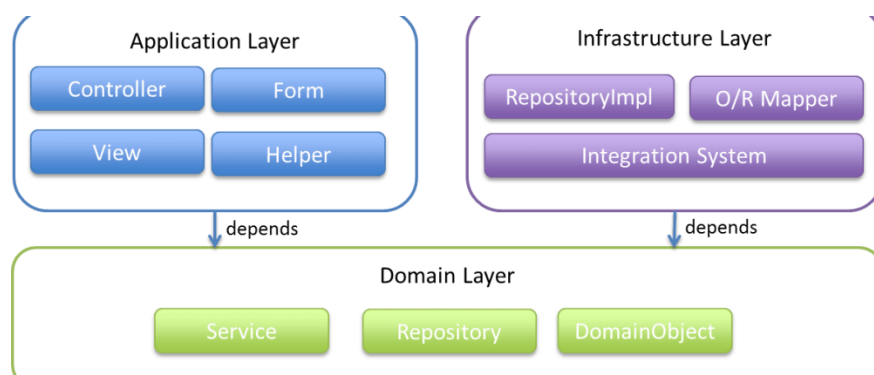
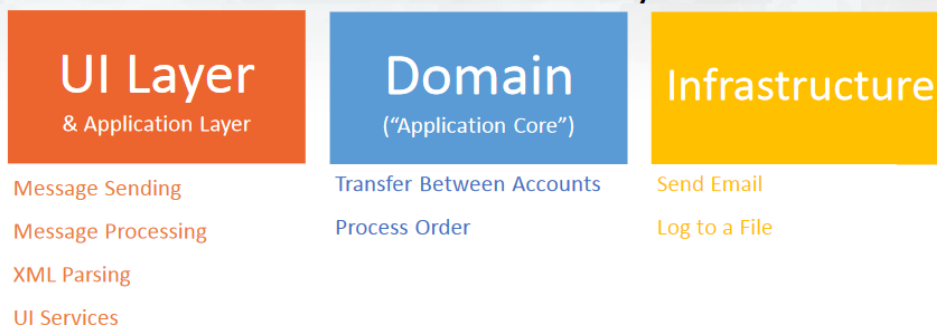
Bounded contexts themselves are an effective way of splitting application ownership across different teams in an organization.

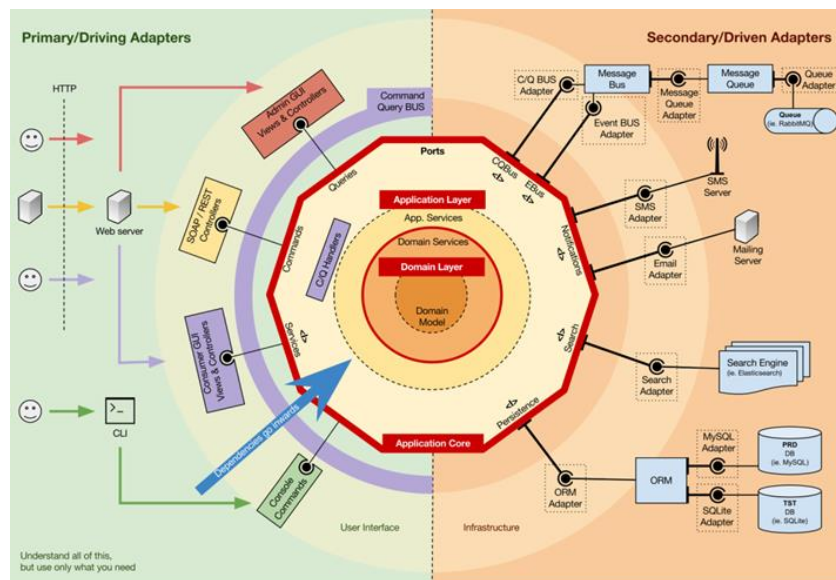
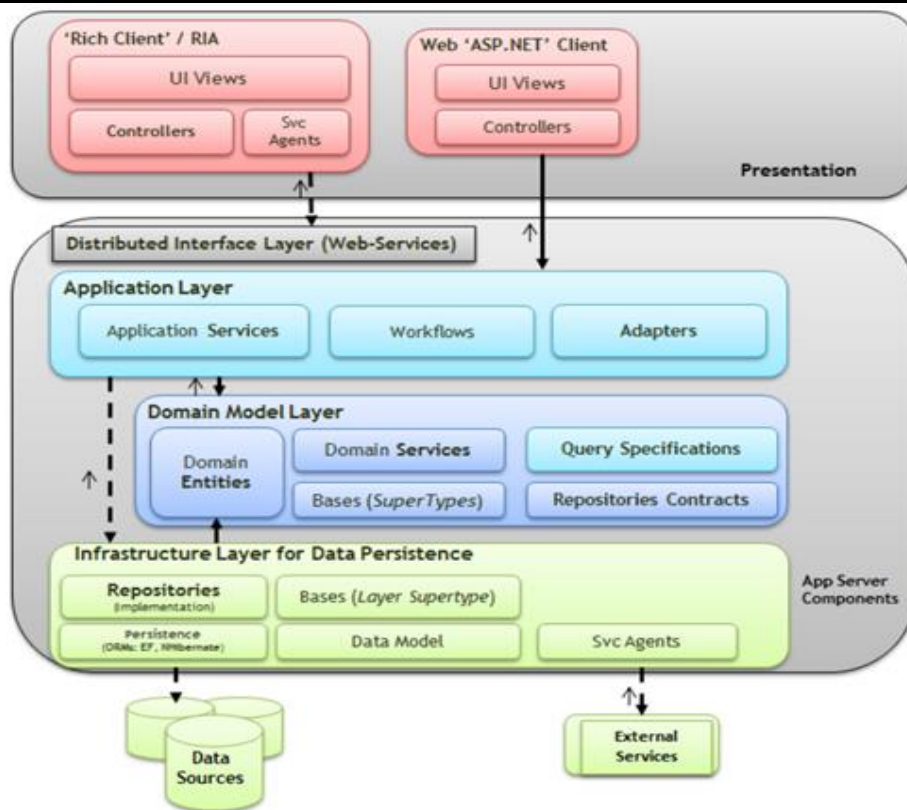
Forming teams that own services in specific bounded contexts takes advantage of the inverse version of Conway's Law : if systems reflect the organizational structure that produces them, you can attain a desirable system architecture by first shaping the structure and responsibilities of your organization.



A possible model of service and capability ownership by different engineering teams as the size of engineering organization grows

Services in Different Layers

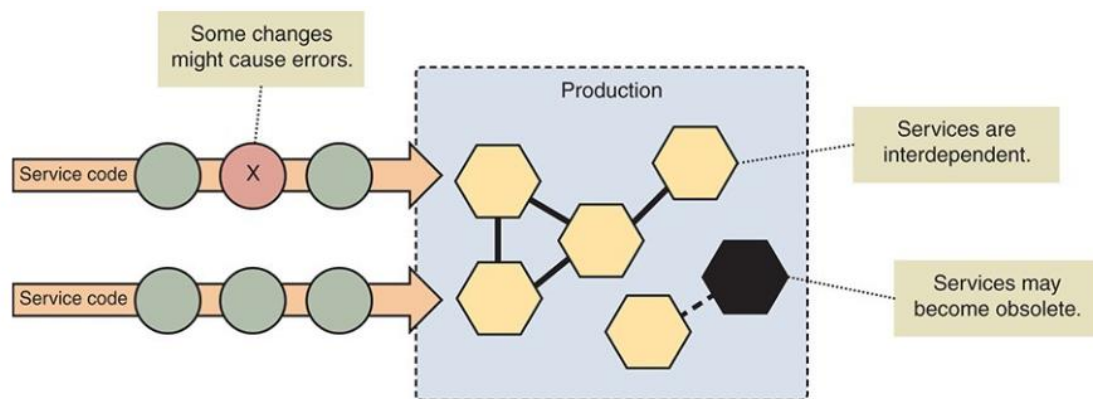




Deploying microservices

Unlike a monolithic application, where you can optimize deployment for a single use case, microservice deployment practices need to scale to multiple services, written in different languages, each with their own dependencies. You need to be able to trust your deployment process to push out new features—and new services—without harming overall availability or introducing critical defects.

Deployment is the riskiest moment in the lifecycle of a software system. The closest real-world equivalent would be changing a tire—except the car is still moving at 100 miles an hour. No company is immune to this risk: for example, Google's site reliability team identified that roughly 70% of outages are due to changes in a live system



A high-level view of production deployment

Microservices drastically increase the number of moving parts in a system, which increases the complexity of deployment.

We face four challenges when deploying microservices ;

Maintaining stability when facing a high volume of releases and component changes

Avoiding tight coupling between components leading to build-or release-time dependencies

Releasing breaking changes to the API of a service, which may negatively impact that service's clients

Retiring services

When you do them well, deployments are based on simplicity and predictability. A consistent build pipeline produces predictable artifacts, which you can apply atomically to a production environment.

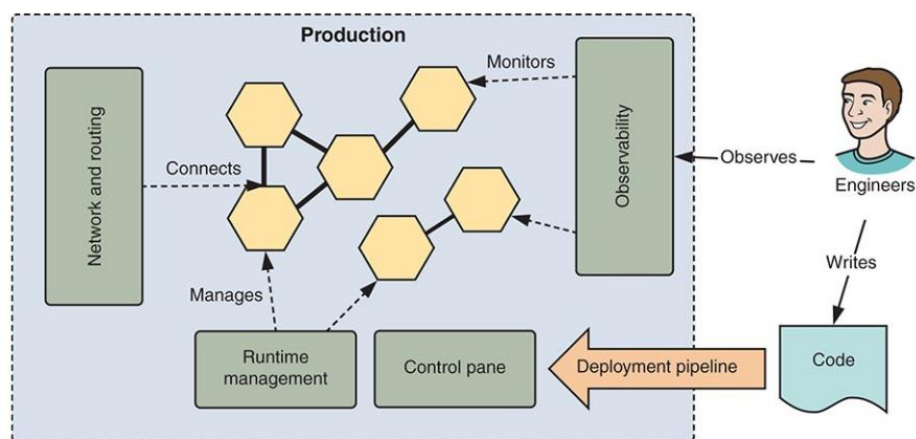
A microservice production environment :

Deployment is a combination of process and architecture :

The process of taking code, making it work, and keeping it working

The architecture of the environment in which the software is operated
Production environments for running microservices vary widely, as do monolith production environments. What's appropriate for your application may depend on your organization's existing infrastructure, technical capabilities, and attitude toward risk, as well as regulatory requirements.

The production environment for a microservice application needs to provide several capabilities to support the smooth operation of multiple services.



A microservice production environment

A microservice production environment has six fundamental capabilities:

1. A deployment target, or runtime platform, where services are run, such as virtual machines
2. Runtime management, such as auto healing and autoscaling, that allows the service environment to respond dynamically to failure or changes in load without human intervention

(For example, if a service instance fails, it should automatically be replaced.)

3. Logging and monitoring to observe service operation and provide insight for engineers into how services are behaving

4. Support for secure operation, such as network controls, secret management, and application hardening

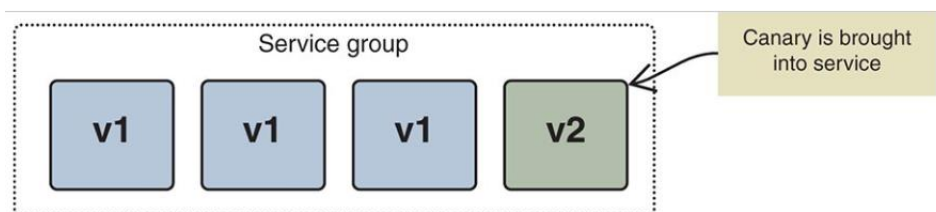
5. Load balancers, DNS, and other routing components to route requests from users and between microservices

6. A deployment pipeline that delivers services from code, safely into operational usage in the production environment. These components are part of the platform layer of the microservice architecture stack

CANARIES AND ROLLING DEPLOYS ON

We can add the canary instance to the group and the backend service to begin receiving requests

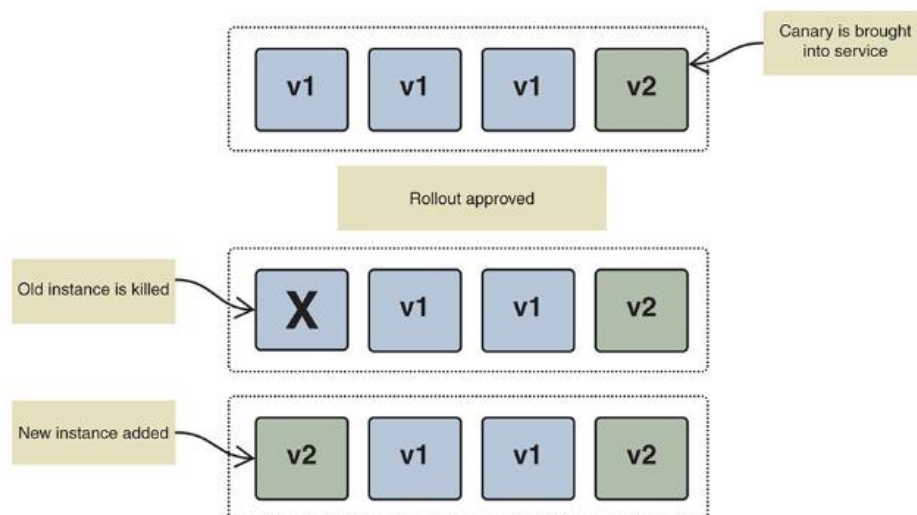
The speed at which this update occurs depends on how much capacity you want to maintain during the rollout. You also can elect to surge beyond your current capacity during rollout to ensure the target number of instances is always maintained. Figure below illustrates the stages of a rollout across three instances



You add a new canary to the group

If you were unhappy, you could roll back the canary.

The command for a rollback is identical to a rollout, but it goes to a previous version. In the real world, rollback may not be atomic. For example, the incorrect operation of new instances may have left data in an inconsistent state, requiring manual intervention and reconciliation. Releasing small change sets and actively monitoring release behavior will limit the occurrence and extent of these scenarios



Stages of a rolling deploy, beginning with a canary instance

Deployment with containers and scheduler

Containers are an elegant abstraction for deploying and running microservices, offering consistent cross-language packaging, application-level isolation, and rapid startup time.

In turn, container schedulers provide a higher level deployment platform for containers by orchestrating and managing the execution of different workloads across a pool of underlying infrastructure resources.

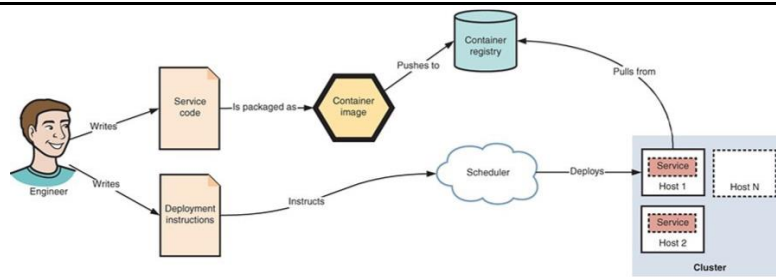
Schedulers also provide (or tightly integrate with) other tools —such as networking, service discovery, load balancing, and configuration management—to deliver a holistic environment for running service-based applications

Docker is the most commonly used container tool, although other container runtimes are available, such as CoreOS's rkt. An active group—the Open Container Initiative—is also working to standardize container specifications.

Some of the popular container schedulers available are Docker Swarm, Kubernetes, and Apache Mesos; different tools and distributions are built on top of those platforms. Of these, Kubernetes, Google's open source container scheduler, has the widest mindshare and has garnered significant implementation support from other organizations, such as Microsoft, and the open source community

We significantly increased deployment velocity using Kubernetes. With Kubernetes, any engineer can now deploy a new service in a few hours. Docker packages service code into a container image, which is stored in a repository. We can use deploy instructions to tell a scheduler to deploy and operate the packaged service on a cluster of underlying hosts.

A successful deployment is about more than running a single instance. For each new version, you want to build an artifact that you can deploy multiple times for redundancy, reliability, and horizontal scaling.



The process of deploying service code to a cluster scheduler

Points of Essence : Within Steps of Phase D : Technology Architecture :

Select Viewpoints

Get ready to meet Stakeholders and think in terms of Technical Context
furthering the Value Chain built through Business Architecture
and the Logical path build through Application and Data Architectures

Viewpoint from needs of Implementation Platforms & technologies;
Framework & runtime tools

Viewpoint from various Infrastructures : Software platforms, Hardware,
Connectivity ...

Viewpoint from Systems and IT Operations Management, Lifecycles and
many NFrs

Select Reference Models, Tools

Where to look for more guidance (from our Repository), What to refer
(rich library of Reference Materials – TRM,)

and what Tool / techniques to use in producing the target result

UML Tools and Other

What is already available as developed Baseline Architecture

Previous project BA, AA, DA and TA of completed versions, artifacts,
current project previous works :

Diagrams, Structures, Execution descriptions

White papers, POC Infrastructure segments

What to develop as Target Architecture

Not entire artifact list of TOGAF; a practical and useful set of Building
Blocks suggested

Some BBs may focus on Enterprise-wide Technology Architecture rather
than being Portfolio Specific

Do not forget that infrastructure is shared across applications and data
systems

Also note that Cloud poses a change in Infrastructure approach

Technology Segment Architect finds the exact Gap of what is needed,
performs by producing Building Blocks

TA

Designing Platform Architecture for App/ Data Components in :

0.2

**Order Placement Scenarios
Order Processing Pipeline**

0.3

**P1 : Detail Order Lookup, Vendor
Registration, Customer OnBoarding ...
P2:
P3 :**

0.4

**P1 : Checkout, Payment, Confirmation, ..
P2 : ??? P3 : ??? P4 : ???**

0.5

**P1 : Credit offers; Multiple Payment modes ..
P2 : P3 : P4 :**

And so on, looking into Cloud enablement where appropriate, interoperability with all existing infrastructures where needed, considering the latest technology in affordable and subscriber solutions

While proceeding in each version slice, covered many Data-level gaps and came out with Building Blocks : Consistency, Accuracy, Completeness based gaps, Transformational and Translation Gaps, Data Ingestion Automation gaps, Movement of Data objects and gaps therein, Concurrency areas, Sync and Async communication with the data stores and event-sourced updates

Building Blocks including Data structures and relationships

Class structures and data access relationships

Fit all work pieces in the defined Business Roadmap

Strategic, Tactical, Continuous change landscape with integration at infrastructure level

To be in tune with long-term Enterprise Landscape

Resolve Impacts : Across Architecture Landscape. From TA angle

Resolution sessions with Business, Application, Data Segments

Is this ok to fit as one or more engineering related segments and components ?

Are these in tune with Business level Views and artifacts ?

Is this information need ok to meet application and Data Component demands ?

Stakeholder Review of these initiatives

Review relevant portions with relevant Stakeholders : as per Stakeholder Matrix

NFRs need focus and discussion with many discrete stakeholders

Agreed ? : Great

Want modification ? : discuss pros and cons

Trade-off between Stakeholders needed ? : escalate through NFR, AOP considerations

Finalized Technology Architecture

At end of every version slice, that much Architecture is finalized and documents (ADD, ARS) updated

EA decides if it is ready for reuse by others (Landscape placement in repository) or not