**Table of Contents:**

**Program Delivery:**

**Purpose:**

* Development team bootstrap?
* Define roles and responsibilities
* Team structure tactic –
* Team composition, Standard team members, their responsibility, how many services per team
* Agile principle – Refer Agile principle
* Microservices team culture
* Sprint 0 Or Spikes --Amey
* Setup workstation

Please refer [**How to Set up the Development Environment**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Workstation_Environment_Setup_Instructions_V1.0.docx?d=w2af22c6ef68944bc81905af9c199ca9d&csf=1&web=1&e=0agAhX)**:** for more details.

* What should be the responsibilities for Development teams per role?

Below are various roles and their responsibilities in detail.

* [**API Designer/Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Designer_Lead_V1.0.docx?d=w73c2348b23f44fc888b61d4eae98e9fd&csf=1&web=1&e=qnsDsy)
* [**API Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Developer_V1.0.docx?d=w6ed53b9636a741c9aa30fd37e1d75967&csf=1&web=1&e=LcQVWm)
* [**API Automation Tester**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Automation_Tester_V1.0.docx?d=w88755d2a824346f1954ad5ff86df85a8&csf=1&web=1&e=6aZIqn)
* [**Cloud-Native Microservices Architect/Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native%20Microservices%20Architect-Lead%20Play%20Book.docx?d=waa9b6e0de053415bb44ade00b2ec63d2&csf=1&web=1&e=P3MOan)
* [**Cloud-Native Microservices Senior Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native_Microservices_Senior_Developer_V1.0.docx?d=w182a421bd4424dde9f7804c8cc55589c&csf=1&web=1&e=F4TQ66)
* [**Cloud-Native Microservices Junior Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native_Microservices_Junior_Developer_V1.0.docx?d=w171c565a5bad4f73ab0df84412217a6f&csf=1&web=1&e=u6KJ5g)
* [**DevSecOps Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/DevSecOps_Lead_V1.docx?d=w78d387a33e5848df8e43762b6a96aa25&csf=1&web=1&e=XMpm2E)
* What are the best practices in development?
* Generic
* CONSIDER use of circuit breaker pattern to CUT-OFF long processing requests
* DO design API for peak efficiency. CONSIDER API end to end processing threshold boundaries from MS (min) to 5 secs (max)
* CONSIDER API async design for long processing requests with notification, rather than using Polling pattern
* CONSIDER using webhooks / callback patterns for Service-to-Service calls
* AVOID tight coupling functionalities to single API. Decouple unique functionalities to separate API and orchestrate API calls to achieve the business goal.
* Consider caching API response
* DO limit data through API calls for better performance. Think of Pagination, limit, offset, seek
* Requirement
* Requirement gathering templates/Checklist –Below is template for Business requirement document  [**Business Requirement Document Template**](https://capgemini.sharepoint.com/:w:/r/sites/CloudCOE989/Shared%20Documents/IP%20-%20Public/Delivery%20Playbook%20-%20Integration%20CU/Mulesoft/Business%20Requirement%20Document%20Template.docx?d=wb1f5120212bc4d19933307fb7090c5c9&csf=1&web=1&e=akoaWq)
* Estimation

1. **T- Shirt sizing the efforts**

At various scenarios of development, the scope and the efforts at the API gateway and integration layer is required to be estimated.

The following format of estimation template is based on T Shirt sizing by identifying a list of estimation objects (The list of items to be estimated) and defining the complexity in categories as simple, medium, and complex.

The quantity or size of work is defined as estimation object \* its quantity categorized into simple, medium, and complex. The size for “Estimation object -2” (e.g. Design) could be 3 simple, 4 medium and 2 complex.

The definition of complexity per estimation object must be elaborated with a unit of effort attached against each complexity category.

Categorizing sizing across estimation objects and complexity as below

|  |  |  |  |
| --- | --- | --- | --- |
| **Estimation objects** | **Simple number of items** | **Medium number of items** | **Complex number of items** |
| Estimation Object -1  E.g. Analysis | **6** | **-** | **4** |
| Estimation Object -2  E.g. Design | **3** | **4** | **2** |
| Estimation Object -3  E.g. Build and Unit Test | **-** | **7** | **3** |

Defining unit of effort across estimation object and complexity category

|  |  |  |  |
| --- | --- | --- | --- |
| **Estimation objects** | **Effort for Simple** | **Effort for Medium** | **Effort for Complex** |
| Estimation Object -1  E.g. Analysis | **1** | **2** | **4** |
| Estimation Object -2  E.g. Design | **1** | **3** | **5** |
| Estimation Object -3  E.g. Build and Unit Test | **1** | **3** | **6** |

Efforts for estimation object -2 = Size \* Unit of effort

I.e. ((3 simple \*1 PD) + (4 medium \* 3 PD) + (2 complex \* 5 PD) = 25 PD

* Designing
* Spring Cloud Config: - It provides support for externalized configuration in a distributed system. With config server we have central place to manage applications across all environments.
* DevSecOps Strategy: - Please refer below link for more details.

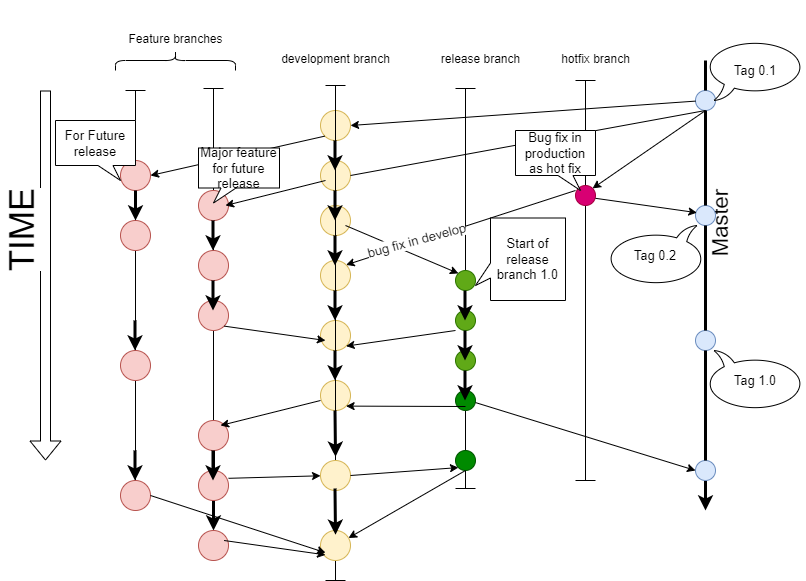
[DevSecOps Playbook](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/DevSecOps_Lead_V1.docx?d=w78d387a33e5848df8e43762b6a96aa25&csf=1&web=1&e=wruPs9).

* Resilient Architecture overview
* NFR Considerations
* Caching
* Coding
* **Code commit:** Include some reference either Jira#, Req# or Defect# whenever you commit the code
* **Constant declaration:** Whenever possible, constants list should be converted to Enums.
* **Std outs:** Printing to standard outs in unit/regression/integration tests should be avoided as the output never shows up in the reporting
* **Package names:** Packages and their names should be in line with DDD principles
* **Log4j:**  Wherever possible avoid having multiple log4j.properties and XML configuration file instead combine them into a single log4j.xml
* **Sample code:** Embed sample code in low level design wherever possible
* Branching: Best practice is to have appropriate branch strategy for e.g.: Bugfix, Feature, Hotfix, Release. Avoid direct check-in into main branch. No check in the release branch after release. These branches must be merged to keep them updated.
* Below is detailed document on Java coding standards and best practices

[Java Coding Standards and Best Practices](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Capgemini%20-%20Java%20Coding%20Standards%20and%20Best%20Practices.docx?d=w3378064236584db389be9c6fdc1dfdac&csf=1&web=1&e=X9eFaa)

* Code review checklist
* Planning
* Code Review
* **Sonar Code Review: -** Define quality gates using sonar to ensure team is not including any blocker or critical.
* **Sonar Lint Plugin: -** Developer can install this in local IDE and run even before committing the code
* Testing
* Source Control and Release planning

Suggested branching strategy is depicted in the below flow diagram:



Basic Rules to be followed:

1. New branch should be based on MASTER only
2. Hotfix branch should be created only for an immediate incident fix. Branch life should not more than 3-4 days. To be used only for changes which can only be tested on PROD.
3. All Feature changes should go via Feature branch only. Release team to mandate the required governance
4. Integration to release branch should go via the Code-Review process
5. Right to merge should only be with **technical lead**
6. Developer branch to be created only if developer needs to work independently of the current feature iteration. Responsibility of the catch-up and merge lies with the developer

* Java Upgrade and Server upgrade

**Best practices for Upgrade projects: -**

* Assessment phase is critical to success of upgrade. Create a wholistic view of existing systems and all interfaces and identify the areas of change and its impact.
* Create a comprehensive list of all the software’s that are changing.
* Understand differences between versions/Technology and take wholistic approach. Blueprint of changes to be prepared.
* Involve Vendors in preparing upgrade strategy.
* Understand communication methods between different entities and impact of the upgrade on them.
* In cases which also involve hardware or middleware upgrade, even though by other teams, impact on needs to be assessed.
* In scenarios where multiple parties are involved, take into consideration – how application development would continue till the time dependent applications or components are delivered by other parties.
* **Server Migration**
* What are the key performance metrics for the success of development team? And how to measure them? (Example defect density, tech debt, velocity, quality)
* Velocity - number of story points per sprint before and after (Industry standard approach on measuring velocity - some reference of tools)

**Below tools can be used: -**

* GIT Burnup and burndown chart overview
* JIRA dashboards overview
* Defect density per release
* Defect leakage across release
* Number of severity defects per release (Code quality defects)
* Number of code review comments per commit / release
* FTR -
* Throughput / improved delivery for a given capacity
* Build and leverage CAP-H accelerator
* **What should be the process and best practice if the project has “API Integration” scope?**
* Please refer [Integration CU Delivery Playbook](https://capgemini.sharepoint.com/:w:/s/CloudCOE989/EdUTuadcxn9IpVyJXpegEtwBIxz8Q2Xu0Um1cT1xjsHghw)
* DDD overview if project has DDD scope Please refer to below link for more details

[Microservices Architecture and DDD](https://capgemini.sharepoint.com/sites/CloudMicroservices/Shared%20Documents/General/3.%20Delivery/Delivery%20Playbooks/Open%20Source/Cloud-Native%20Microservices%20Architect-Lead%20Play%20Book.docx)

* Capgemini’s proven integration patterns (Reference Architecture) and POVs
* Reference Architecture --Deepak, Amey

Please find below Reference Architecture and Maturity model for Microservices

A black background with a white and red logo

Description automatically generated

* API Gateway
* Service Mesh (Security, Traffic Management and Observability)
* EDA (Security, Traffic Management and Observability)
* M2M
* GraphQL
* Tools and Accelerators --Deepak

**Overview of Accelerated API & Microservices Development -**

* 6 Steps Process covering from Cradle2Grave

1. Domain Driven Design to understand the complex business domain concepts and map it into software artifacts
2. Estimate for the underlying API & Microservices Development work
3. BDD Approach for Automated Acceptance testing.
4. API first approach with Contract based Testing for Rapid API Development
5. Microservices framework to jumpstart development
6. Containerized CICD for deploying it on Cloud PaaS

* Describe about the DDD exercise covering
* Domain Discovery.
* Strategic Design with domain, subdomains & bounded context.
* Context Map for teams to collaborate and integrate software.
* Tactical Low-level Design.
* Visual Modelling.
* Identify APIs from the Low-level Design and Visual Modelling
* Tools & Accelerators – DDD Toolkit
* Describe about the Estimation Model and Techniques covering
* Estimation Complexity Matrix
* T-Shirt Sizing, Story Points
* Tools & Models. – API Estimation Matrix
* API First Development Approach covering Lifecycle
* Design
* Test & Build
* Deploy & Publish
* Subscribe, Consume & Feedback
* Fix, Extend & Version
* Deprecate & Retire
* Accelerators. – BDD Swift, API Center,
* Microservices Development
* Designing Microservices
* Data Management
* Inter Services Communication
* Building, Testing, Deploying & Running
* Monitoring & Observability
* Accelerators. – uServices Express, OmniCloud
* DevsecOps for Continuous Delivery
* Containerized CICD Pipeline
* Integrated SecOps
* Continuous Feedback

**Product Based Delivery –** Below are the links for activities like setting development environment, responsibilities for various roles.

[**How to Set up the Development Environment**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Workstation_Environment_Setup_Instructions_V1.0.docx?d=w2af22c6ef68944bc81905af9c199ca9d&csf=1&web=1&e=0agAhX)**:**

* Description of Tool stack (Eclipse, IntelliJ, STS, SOAP-UI, Postman, SWAGGER/RAML)
* Pre-requisites for workstation
* GitHub Access and Permission Set up
* Repository Set up
* Branch Set up
* GitHub Notification Set up
* Postman Set up
* SOAP-UI
* Greenlight Set up in IDE

**What Task Do I need to perform for my Role:**

* [**API Designer/Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Designer_Lead_V1.0.docx?d=w73c2348b23f44fc888b61d4eae98e9fd&csf=1&web=1&e=qnsDsy)
* Define API Taxonomy
* Create API Spec/Schema
* Define Endpoints
* Define Parameters
* Define Request bodies
* Plan sprints and identify story points inside each sprint. Identify scenarios that make up story points.
* Write Gherkin scripts for the scenarios
* Define Metrics for API Usage
* [**API Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Developer_V1.0.docx?d=w6ed53b9636a741c9aa30fd37e1d75967&csf=1&web=1&e=LcQVWm)
* Write acceptance test scripts from gherkin skeleton
* Write unit tests to test code functionality
* Iterate to build out sandbox implementation
* Repeat until release candidate
* Throughout this process use Github to version & manage code
* Import Swagger definition to Publisher, to create base API.
* Update API configuration with backend details
* Update API with throttle, security & error policies.
* Update any additional documentation required
* Publish API to store (as per need define additional constrains during publish)
* [**API Automation Tester**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/API_Automation_Tester_V1.0.docx?d=w88755d2a824346f1954ad5ff86df85a8&csf=1&web=1&e=6aZIqn) -
* Developing the Contract Tests Scripts
* Setting Up Contract Tests in Postman
* Adding Test to a Contract Test Collection
* Running & Fixing Contract Tests
* Diagnosing Contract Test Failure
* Setting up Mock Servers
* Testing with a Mock Server
* Developing Test Data
* [**Cloud-Native Microservices Architect/Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native%20Microservices%20Architect-Lead%20Play%20Book.docx?d=waa9b6e0de053415bb44ade00b2ec63d2&csf=1&web=1&e=P3MOan)
* Identify the Bounded Context and Design the Microservices Inner Architecture
* Design the Data Model for the microservice
* Design the Integration/Collaboration of microservice
* Design the NFR’s
* Build & Unit Test
* Process Pull requests
* Perform Code Review
* Deploy on Higher Environment and verify E2E tests results
* [**Cloud-Native Microservices Senior Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native_Microservices_Senior_Developer_V1.0.docx?d=w182a421bd4424dde9f7804c8cc55589c&csf=1&web=1&e=F4TQ66)
* Code Spring Boot and Spring Cloud Services
* Code & Unit Test the Eventual Consistent transactions
* Unit Test and Deploy on Dev
* Debug issues in Inter Services Communication
* Troubleshoot issues for failed tests
* Perform Peer Code Review
* Validate and Fix Builds and Build failure
* Promote Code onto higher environment
* [**Cloud-Native Microservices Junior Developer**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/Cloud-Native_Microservices_Junior_Developer_V1.0.docx?d=w171c565a5bad4f73ab0df84412217a6f&csf=1&web=1&e=u6KJ5g)
* Code Spring Boot and Spring Cloud Services
* Code & Unit Test the Eventual Consistent transactions
* Unit Test and Deploy on Dev
* Debug issues in Inter Services Communication
* [**DevSecOps Lead**](https://capgemini.sharepoint.com/:w:/r/sites/CloudMicroservices/Shared%20Documents/Delivery%20Play%20Book/Open%20Source/DevSecOps_Lead_V1.docx?d=w78d387a33e5848df8e43762b6a96aa25&csf=1&web=1&e=XMpm2E)-
* Set Traceability
* Create Feature Branch
* Create Release Branch
* Remove Release branch
* Remove Feature Branch
* Activate email log on builds
* Onboard Application onto CICD Pipeline
* Publish the various Reports – Deployment, Release etc.

**FAQ’s –**

1. Is there a publish task checklist available, which lists all tasks that API developer needs to have to publish an API.
2. How do I resolve conflicts in the below cases
3. Synchronization conflicts during development
4. Merge Conflicts during code propagation
5. Is there a best practice available for API Response Handling, which we can refer to
6. Is there a published list of API Metrics Usage?
7. How do I multi throttle a given API considering the system capacity and internal calls.
8. Is there an API Documentation Best Practice available?
9. Is there a guideline or best practice available for RESTful design
10. Is there a best practice available for implementing API Security
11. How can we implement Event Based Microservices for our use case.?
12. I am not able to troubleshoot the failure in my CICD pipeline, I am stuck and not able to proceed, what can I do.

**Team structure tactic**

Agile team structure at the minimum should have the following split up

Suggested Org Chart

A diagram of a company

Description automatically generated

**DevOps team (common across different feature teams)**

* Responsible for continuous deployment
* Creating Jenkins pipeline
* Environment provisioning
* Release management
* Defining branching, release strategies

**QA/Test teams (common across different feature teams)**

* Responsible for continuous Testing
* Cross functional team
* Ensure test strategies/approach are kept consistent across all team
* Test planning and executing based on the release plans
* Test lead will report into DevOps team lead
* Will participate in daily stand ups and work closely with development team

**Development team**

* Responsible for continuous Integration
* Ideally split into multiple Feature team
* Each Feature team will follow a Benzene model having 6 members for each sprint
* Each team will have a Scrum Master who will act as a facilitator and coordinator role
* Product/Feature owner – Business analyst role
* 2 developer roles – Senior and junior ideally full stack developers
* 1 team lead role – play role of a Solution Architect or Tech lead
* Tester role – test resource will be embedded in the team but sit in cross functional test team

Management and Support

* Responsible for Project management across various teams above
* Delivery management
* Client communication
* Management reporting

**Agile principle**

* Agile Team Metrics are powerful tools, which measure the health of the team, team performance, feature development progress. It can be split into
* Sprint metrics

At the end of each sprint, the team can collect sprint metrics in terms of

* UJs planned v/s UJs delivered – measure planning effectiveness/efficiency
* UJs delivered v/s UJs accepted – measure quality of deliverables
* Sprint Burndown

Use to track the product development effort remaining in a sprint

* Release burndown

Release (Epic) burndown charts track the progress of the entire cross functional team until the feature is live

* Team velocity
* Velocity is average amount of work a feature team completes during a sprint measured in either a UJs or hours
* Useful to plan the backlog effectively

**NFR considerations**

|  |  |  |
| --- | --- | --- |
| Category | High level requirement | Brief description |
| Availability | Availability target based on Tier | Tier 1 – Availability target >=99.9%  Tier 2 – Availability target >= 98.5%  Tier 3 – Availability target >= 95% |
| Availability got different periods | Availability as per   * Specific days/range of days * Time zones * Peak hours * Business critical periods * Downtime availability for systems maintenance and housekeeping etc. |
| Impact to availability if the interfacing system is downgraded or unavailable |  |
| How long can the business tolerate an outage/degradation of service | Should identify critical business periods/service hours and availability for the system used |
| Back up | How long do the backups to be kept |  |
| When can the backups be taken |  |
| Process of DB database | Whether synchronous or asynchronous back up is needed i.e. whether full or eventual consistency requirement |
| Predicted volumes of data to be backed up |  |
| Max permitted time to recover data after a failure |  |
| Scalability | Is there a requirement to scale the solution beyond the current forecasted growth |  |
| Security | Details on encryption of data at rest | Eg xxx bit encryption |
| For each message/communication type, what non-repudiation methods are being implemented |  |
| What software vulnerability scanning to be used |  |
| Performance | Expected e2e response time for user transactions | Detail on 50th, 95th and 100th percentile targets for fully rendered responses to user transactions |
| Expected e2e throughput for user transactions | Detail the average(mean), 95th and 100th percentile targets for throughput ( tps/ message per second) |
| Monitoring and alerting | Outline monitoring and alerting requirements in terms of capacity management | Monitoring of:   * E2E transaction Performance and throughput * Service end point * DB query performance * External services |
| Disaster Recovery | Max permitted time to restore the service following DR | How long the application can run with a reduced service level |
| In the event of disaster, detail the minimum level of service | For eg: 50% of users with 100% functionality |
| Recovery time Objective (RTO) for the service |  |
| Recovery point Objective (RPO) with the permitted data loss duration |  |
| Capacity | Business user transaction volumes | Expected volumetrics for next 3-5 years for infra capacity |
| Concurrent users | Peak concurrent users during normal operations |
| User journey wise expected transactions per day |  |

**Requirement gathering templates:**

The below describes the **functional and operational requirements** gathering template

**User’s type** – Business Users, Customers, Operations personnel etc.

**Template 1** – Jira template

**\*\* Background\*\***

<Please provide detailed background of the project requirements here>

**\*\*Requirements\*\***

As a User:

I want to - <Please provide the detailed requirements here>

So that - <Please provide the detailed expectations from the requirement above>

You can add as many requirements /expectations as possible here

**\*\*Acceptance Criteria\*\***

Given - <Please add your test case criteria here>

When - <Please add your test case scenario here>

Then - <Please add your test case expectation here>

You can add as many acceptance criteria as possible here

**\*\*Value Add\*\***

<Please add/list all the value adds to the customer on this requirement>

Template 2 – Generic

**\*\*Background\*\***

<Please provide detailed background of the project requirements here>

**\*\*Scope\*\***

<Please provide detailed scope of the project requirements here>

**\*\*AS-IS\*\***

<Please provide AS-IS behavior of the project requirements here>

**\*\*TO BE\*\***

<Please provide the expected behavior of the project requirements here>

Resilient Framework overview

