How to Scale 100+ MuleSoft APIs as a Platform Owner using AI.

No fluff guide to scale Anypoint Platform pass 100 APIs in the AI Era

Ngoc Quang Nguyen

MuleSoft Integration Specialist

How to Scale 100+ MuleSoft APIs as a Platform Owner using AI.

No fluff guide to scale Anypoint Platform pass 100 APIs Solo in the AI Era

Ngoc Quang Nguyen
MuleSoft Integration Specialist
2026 Edition

Contents

Chapter 1

Introduction

Most MuleSoft implementations failed spectacularly to get user adoption, so hard to maintain and so expensive that companies who are using them constantly want to move to other platforms.

While the Consultant and Enterprise Architect thumping their chest and congratulate themselves on the new roll out of Mulesoft. While internal devs scramble to keep the broken APIs running

MuleSoft Developers are so in demand that Devs can choose to leave whenever the messy platform get too overwhelm or if they stay they would be so burnt out from maintaining the platform that it's impossible to grow beyond 20-30 APIs

Most company solution is too hire more Devs and Architect. But it's hard to change way of working or do a refactoring when the platform is messy and broken.

With Agents it's now possible to use Mulesoft with minimal resources. No need for a team of 10 to maintain 100 APIs. Fewer people meaning team is easier to manage, knowledges are easier to transfer. Agents takes care of all the small bugs, typos, human errors, team can now focus on make the platform thrive and grow.

This book was first written down as a condense of my 6 years of experience as MuleSoft Specialist. At the time of writing this book Claude Code and Opus 4.6 came into the scene which force me to rethink the way we work with MuleSoft. This book was then adapted to solve the problem of using AI to build and maintain a platform with 1 Mulesoft Specialist.

This goal challenged me to change my old way of working trying to squeeze the most juice out of my brain and AI tokens. I had to become extrem agil and challenge my team to be as async as possible.

If you are anti-AI coding. This book might not be for you.

In this book I will guide you through how I build and maintain a platform of more than a hundred APIs alone.

Chapter 2:

From 0 to 10 APIs — Quick Wins and Why Getting Stakeholder Trust Is Make or Break of Your Platform

A big mistake of Mulesoft Platform Owner or Mulesoft Specialist that responsible for Mulesoft is thinking after their company bought Mulesoft it's done. Stakeholders have to use it no matter what. The truth is Mulesoft is not a one time purchase but an expensive subscription. So your stakeholders will constantly revaluate it usage.

As the one who responsible for the platform you (especially solo) will also be evaluate. And the question usually is are there any working integration? And is it useful? No matter how nice your powerpoint report and how cool your Proof of Concenpt Presentation is. Those will always be the questions in the back of the stakeholders or users mind.

That is why the goal for this stage is get as many working integration out as possible. Not setting the foundation, not CI/CD pipeline, not standardised documentation, not RAML library, not CDM…

Although setting the right mindset is still important for the success of the platform at this stage.

This section will deal with a few key mindsets and then get you development enviroment set up and shipping integration as soon as possible.

Which early mindset trap makes a MuleSoft implementation fail or become impossible to scale?

People often think that lack of experience or expertise is what make project fail but it's actually mindset is the key to success or failure of a project. Mulesoft projects used to be so big and complex, many moving pieces, many people involved no one no how to manage it and in the end no one know what went wrong. Easy to just blame it on expertise or experience.

But Mulesoft is pretty simple to build and it's quite hard to fail. There are many guardrails and it quite easy to fix or rebuild especially now with AI Coding. But most project still fail why? It's all because of mindset. I remember reading the quote somewhere that everything that was built in the real world has to be first be imagined in your mind.

If you've ever been to a Mulesoft project or any IT project you know that most was discuss was many risks and pitfalls of how it will fail. People starting to outsource their responsibilities to external factors. The degree is vary but I have never been to a project that have a clear vision of the platform.

But now is the chance to do that, by building solo you avoid the risk of bringing in people that undermined your vision. Don't get me wrong I love working with people but it's also true that it's very hard to work with people because people are not cork in a machine, they have their own thoughts and

mindset that you will have to deal with. If you bring in the right people, your platform can grow exponentially faster and in reverse if you bring in the wrong people it can destroy your platform.

This is why big name with all their experience and expertise like Delloite, Accenture,… always make mess that company then to go in and clean it up after to be able to use it. They try to bring in the many experts right from the start with conflicting vision and mindsets. So no matter how experience they are it's impossible to realize multiple visions at the sametime on 1 platform.

By building solo with AI not only you can realize a single vision in your mind that make the platform cohesive and stable. You can overcome many challenges just with your imagination. And of course you can move 10 times faster by yourself. It's also better for other stakeholders because you are the only one that responsible for the platform so they can all come to you when things when wrong. Unlike traditional team where you have to go through 2 3 levels before you find the one who write the codes.

Now let's dive deep into a few key mindset to have or to fix so that you can realize my vision of a solo Mulesoft Specialist that take care of a platform of hundred of APIs alone.

"Ship fast" or "best practices"?

One of the thing I really got me thinking about best pratice is when I start getting into indie hacker community. I read story about founders who run a platform of thousands of users with 100k of monthly revenue on just 1 php index file, or who don't know how indexing work,… The list go one and one.

That got me thinking, we are here follow the "best practices" of company like Microsoft, Adobe or Salesforce with horrible to meh products, while ourselves struggling to ship an integration with few hundred of users. Is it really the best way to do this?

- One metaphor that I like is imagining you putting a key inside a key hole. You learning all these rule, your arm has to bend 45 degree up and your wrist 90 degree to the ground and you try your best not to touch the side and put it in perfectly. What will happen is your hand ended up shaking unconstrollbaly.
- Same thing happen when you applied 10 best practices on top of already constraint platform and systems that you have to integrate, this can paralize your vision of the.
- Let's think about the worst practice ever is to commit your password in the repository, unencrypted. How often do you make your company repository public? Even if it is public it take a few seconds to deactivate or change the password and remove it from the repository.
- Not to say that you should do this but actively reduce and simplify your process and commit a crime of "bad practices" to ship faster is most of the time fine. And you will have your AI companion that will help you implement most of the important stuff with best practices anyway.
- So in conclusion, ship as fast as you can and don't worry too much about best practices.
- Still in these book I will show you some best practice which are absolutely necessary and some which are trap that will suck all your resources for nothing.

Build and rebuild

- Usually I see a lot of junior Mulesoft Engineer made this mistake of not set the time for refactor and rebuild when building a new API. They try to build it perfect right at the first time.

- This sounds contradicting but when you are afraid of errors and bugs, there will be more errors and bugs happen. Senior Engineer build better APIs not because their knowledge of all the theoretical bugs and errors, but because they build so many integrations, APIs that it's their muscle memory. They're not trying to prevent all the bugs but they know this component here will have this bug or this error so they just naturally build with that in mind.

- While junior will read about all the bugs and best practices but have no experience of how to apply them in real project. Nuances matter.

- That's why build ship and fix or even rebuild after not only teach you all the real experiences, And when you have time to refactor and add necessary efficiencies, you are much more confident and relax to build. It will make you ship faster instead of overthinking and over optimized. Which make your APIs even better even the first version.

- This is actually the key feature of Mulesoft that people usually missed that let you build fast test and rebuild quickly. All the guardrails were built-in so critical failures rarely happen.

You are a service provider

- Most engineer when get into Mulesoft integration shock to realize that their job 80% like a service provider: collecting requirement, connecting team together, mapping, testing with stakeholder,… And not just build API.

- But if you can understand this dynamic you can flip the script and have other team come to you with prepared requirements to ask to you to provide integration and ready to do whatever it takes.

- If you are the platform owner or team leader, do not fall into the trap of letting your team blame the other departments for lack of cooporation, understanding of data model. They are technically your users/clients eventhough they aren't the one who paying you. And love it or hate it you are providing a service to them. US teams do this very well compare to European teams. Maybe it's in their education?

- In most team I worked in, people always try to run it like a IT department. The truth is that model will lead to fail. You probably know how IT department usually preceived, a bunch of know it all that can't get anything to work and usually break things.

- You should always run it like a in house consultancy. The problem with external consultancy is they don't really have a stake long term. Even if they try their best they won't be able to persist or be patient when thing got difficult. But companies needs them cause they know how to provide a service, treats other departments like clients, they don't just say. They know how to guide, consult, deal with people. Which in turn result in better integration built.

- Think about when you in a restaurant you ask for a recommendation and the waiter said I don't know what you like, what would you think? A simple question you will understand if it's a good restaurant or not.

- I remembered in one of the project I worked with, the team is pure internal, have expertise, good structures and processes in place, but other departments were complaining about them not delivering. Simply because they were too rigid and don't have servicing skill.

Why take control of other systems is important?

- The things is integration is such a black box sometimes to even devs, even if they know it's some kind of connect to other system. They still don't know about other system or how the API work underneath. Most Devs work in a system usually lock in the front ends, maybe sometimes some automation in the backend. They work too long and go too deep in their own system that it's hard for their brain to switch context.

- I remembered when I delivered a SAP integration in one of the project, a SAP consultant joking that it some kind of magic since they couldn't get that to work for a while.

- That's why taking control or get visibility for the systems that you integrate is not burden but advantage. It will save you huge headache in the future.

- In one of my project, the team strictly avoid touching the systems they integrate because they don't want to add more to their workload, which understandable but it ends up cost them huge amount of time since they became reactive instead of proactive to other systems changes.

- Myself recently also experience this. We spent a huge amount to debug time for SAP cause I didn't get the access to SAP directly and route the testing to other people. This creates blind spots and huge frictions when debug because I couldn't see everything that happens. Luckily we found out the problem of padding 0 for SAP ID. If I have the full visual on other system I would have solve this in few hours.

- Another points people neglect is when you have information you will have power over other system. Which give you much more control on other teams as well. Like it or not if you work in a big team you will have to get involve with politics, especially with integration, as I call it you have to integrate people as well.

- One way Mulesoft recommend to solve this friction is to deploy Mulesoft Developer to each department and embed them into that team. Even then you as a Mulesoft Engineer you can't still need to take control for reason meantion above, of course this time will be somewhat simpler.

- This the key to success as a Solo Mule Specialist. Because when you are solo, frictions scale fast. By log in and take some responsibility for that system you able to control both side of the integration which also scale fast as you have the full vision on the platform. Now we have the AI so we don't have to spend months to just learn thoese systems but can also ship integrations at the same time. That's why Solo is not only possible but huge advantage if you know how to set it up.

How flexible should you be?

- Often I see in the Mulesoft space where people are always looking to stadardize stuff, building rigid process etc. Yes it's true that you need a process and boundary otherwise the stakeholder will just request one change after another any you will burn out very soon.

- But things are different now with AI, you can make most changes with a few prompts. Does that means you should accept any changes? No, this comes with experience, there are changes that are necessary for the project to run smoothly, there are changes that can be put in the backlog and there are changes that is unnecessary.

- People said in the future engineer will be making the most decision and AI will build. That's very wrong. If we talk about making a decision based on available information AI beat human 100% of the time. Look at chess AI, no human can beat them. But the different is human can make decision based on unavailable information, purely on instinct.

- This meant the more flexible your process is the easier it is to try out decision and pick the best one. How do you train your instinct you experiment a lot.

Why early trust from stakeholders and users is more important than "getting it right"?

- If you follow all the mindsets above it's easy to build up trust with stakeholders. You can kind of see what I'm getting at, the principal is the stakeholders or users won't ever see your Mule Codes, they will never open the Anypoint Studio, they only care about if the integration work correctly.

- People build whatever they think is best practice and and force the users to use it by "training" or "on-board". Since the users are not the one who pay their salaries, it become a war to decide who has the most knowledges and who got the "best" practice.

- The problem with this is you can force people to do what they hate. The users will make your life hell with change request, bugs reports, complains,… Instead of useful input that carry the platform to sucess like they actual have a stake in this.

- This is why we want to build simple useful integrations and APIs in this stage and with only the users in mind. No matter if we have to compromise some best practices we want to do it fast and solve biggest pain point as fast as possible to get them to invest into the platform.

- Similar point as I shared in the mindset section about share responsibility is this will help you have the users on your side and have massive bird eyes on your system landscapes which not only increase the speed of implementation but also the quality of your APIs cause you will have less blind spots. This is key if you ever want to get over 100 APIs and more.

- Then after you can start do thing properly with good information so you can scale pass 50 APIs by yourself. This is very simple with AI.

Gather requirements — Why most projects struggle?

People think Mulesoft or Integration is about coding and engineering but the truth is most of the time you will have to collaborate or integrate different people using epic, features, user stories, bugs. You have to make sure the info that you have is correct and adequate. Since it's involved multiple system with psuedo users like managers that rarely use the system, tester, admin who don't actually use the system for work, or user that not really care to improve the system and just want to get their job done and log off. It's not usually straight forward to pinpoint the problems or usecases.

And most of the time you get users complain is because improper gathering of requirement. You integrate the wrong fields, or the important fields is not proper documented and implemented. This is the most resources consuming part and AI can't do much for you at the moment.

Eventhough users have to use your APIs, and usually there is no worry about user's adoption or competitions. But if you build without the service provider mindset, don't care about real workflow, overengineering, you will still get complaining, constantly change request that will break your APIs, burned you out. A lot of Mule Team got even got phased out as ROI not justified.

The key principal that I use is chopped it up into actual user stories, not just imagine user stories. Actual documented of a workflow. I often see people chopped it up into separate CRUD operation. The problem is it's often too big and unecessary. Like for a story you need maybe 30% of the fields filled and the create operation but you built the full data object with the create flow and then you need to take care of all the user stories, that come with different check boxes, picklist,… Which can involve multiple userstories, departments, systems. This often overwhelm junior architect, and where most project stuck on the loop of missing info and decision.

Focus on a clear goal is key when dealing with multiple moving parts. And when completing multiple small goals you can build momentum really fast, which is key if you want to scale the platform over 100 APIs

Second key is let a Developer do the requirement gathering or do it yourself if you are solo, don't outsource this parth to a Business Analyst or Project Manager.

A developer that understand business usecases can 10x the speed and quality of the development.

Most developer now aday build without seeing the prespective of business this is not only inefficient but dangerous. If you ever seen these experiment scientific cars you understand what I meant, these cars often look ridiculous and uncomfortable to sit in.

I'm not against discussing or brainstorming solution but Devs should always see usecases and have direct contact with users. The problem with integration is if miscommunicate through channel and middle man scale up really fast since it involves multiple systems and usecases. That's why devs need direct channels into users. Other stakeholder can focus on review the outputs and brainstorm solution design.

Common mistakes when gathering user stories in MuleSoft is over reliant on form and questionaires. People just slam 10 templates, mapping sheets in the user's face, ask them to fill these in and it will take forever if it happens at all. Think about when you have to go the the goverment office and fill out a form, how much you hate it even though it's your data that you can pull straight from your head.

How the process should be:

Raw user input:

- In my opinion this is the most important piece that a lot of time got forgotten.
- We are too focus on writing the features that we often missed the neuonces and pitfalls from raw data, this leads to incompleted test cases, unnecessary features, missing high ROI …
- Especially now with AI sometimes you will just need raw input to complete an integration.
- I like to use human languages, tell them to describe their process in simpliest form, tell them to record their workflow. Rappor is also important, it makes the difference between mediocre user story vs a great user story.
- I prefer user input in writing that way users can take their time and think about what they write instead of listening to a lot of noise or don't have enough information cause they don't like to talk. When direct demontration in the system is needed a meeting can be set up.
- A questionaire is usually useful to guide the user but be careful not to be too rigid and guide users to the wrong feature or focus. I like to have different questionaires for different systems or roles.
- After you capture the input, start designing a solution, if a decision can't be reach with the design then you can build a Proof of Concentp and show to the users.
- Try to get a connection to the systems.
- After you have a poc or rough solution classify the problem into an Epic, a Feature, a Story or a Bug Ticket. I like to put most of my requirement into story. Unless is a big requirements that require more than a few days to build, which is rare now with AI.

How to build a Proof of Concept:

It's rare that you only have production system and no sandbox to test. It's always risky to not have a test system but usually you can push a few test data in there for review. But if even that is not the case then the best you can do is imagine that you have one looking at the production and describe how the data would look for the user with screenshots and the likes

User stories:

- A slice of the requirement that can be implement in a few days and clearly test by the user end to end.
- This is where you document the triggering action of the user, the technical solution and the desirable result from the target system from the user's POV.
- It's possible to document it in freeform following those 3 points.
- If it's really big story you can change it into a feature or an epic.

- After you have the requirement and start creating your stories and working on the data mappping sheet. Hard to pin point the perfect time to start but I like to do all 3 in parrallel. That saves me times and help me refine the stories and mapping immediately as we go.
- Don't worry if your story is too big or too vague, you can always refine it.
- I also like to start implement or prototype as soon as I have enough infomation and get feedbacks instead of trying to perfect my story or mapping.
- You will see a lot more nuances and hidden traps of the user story if you always build it in parallel.

If the story is big you can also classify it into Epic or Feature, especially if you have multiple devs or need to collaborate with big teams.

Epic:

- Big high level solution, often take months to build
- Often look something like: "Sales Reps waste time on putting orders from Salesforce into Netsuite"
- Break them down into features that can be built in a few weeks.

Feature:

- Smaller part of the Epic or a smaller requirement. Often take weeks to build
- For example something like: "Sync new Salesforce Orders into Netsuite"
- Break it down into User Stories.

How to build mapping Sheet:

- Mapping Sheet is the most important artifact for Mulesoft. It's the source of truth for communication especially when you are solo.
- Keep your mapping sheet as simple as possible: Source System and Target System (or CDM if you are using one)
- Use seprate sheet for different mapping, try not to use tab since it can be confused when communicating.
- Have columns for labels, required or not, types, and mapping rule in human language (don't write in cryptic or programmatic form even if it means for developers).
- Requirement fields should be highlight and be the focus for the first iterations so you can immediately start with your prototyping.

- Afterward, when the users see the data in their own systems, they can identify important fields much easier, this is when you start the next iteration

Mule super hacker workstation set up for coding

“Why do you have a workstation at all? Just have agent code bro” That’s what the AI Influencer would tell you. Surprisingly enough agents can hallucinate sometimes, although it’s getting better and better. Although I will guide you to how to set up all the agents you need, to be honest with Mulesoft we still need a lot of hand-on, review and monitoring, 90% of the tasks can be done with Claude Code within a few prompts. And it’s will take quite sometimes to set up and have proper Mulesoft agent workflow especially if you are in an large enteprise and aim to scale above 100 APIs

Remember those hacker scenes in the movie where they type in the terminal and then window just pop ups, now you can do the same as a MuleSoft Dev, no more goofy eclipse editor.

It’s actually has been the biggest sticking point of Mulesoft. I remember the meme when you open Studio, get a coffee, open your project, deploy the app, open postman send the request, scroll through the log to find what you need, not to mention the deployment process, performance bugs, production bugs. The bigger the code the worse the process will be. This is the biggest point where it’s impossible to scale Mulesoft. It’s simply too costly to make changes, maintain,… Most platform reach its limit at like 50-80 APIs.

They try to build alternative but Anypoint Code Builder and Flow Design in Design Center is horrendous. So don’t waste your time on it.

In this section I will show you the new work flow with AI Coding, where AI will do the whole local workflow for you without you needing to open Anypoint Studio.

Should you use macbook or window for Mulesoft developement?

- Personally I would take a macbook since they have the unified RAM in the laptops, RAM is the most important of Mulesoft development in a laptop, it help you deploy faster and debug smoother

- Macos is optimized for their own chip and hardware which make anything run much smoother.

- They also built much better. I had a macbook pro 14 inches with 16 gb of ram for 4 years and had no problem and only switch recently as the unified came out. While I used window laptop during my uni time and had to change every 2-3 years cause it became unusable.

- With window you can get a thinkpad with 32gb ram less then 1k. And most company is embedded in Microsoft Ecosystem so usually much more convinience to setup. And Anypoint Studio has way less bugs on Window than MacOs. But for the workflow in this book a Mac will work much better

My Mulesoft AI Coding Work Flow:

- For AI Coding I use Claude Code. The 90$ subscription is enough for me to work (a client with 80 APIs and some part time work). A lot of people use Claude Code inside VSCode but I find that very distractive. I want to focus on just one window at the time.

- For commit code I use GitHub Desktop, I find this is the easy to read changes and easy to do complex commit.

- I only use Anypoint Studio checking the flow sometimes and do some commits history manipulation like reset and force push (you can ignore this for now).

- VSCode for reading files and manually editing xml code

- Maven: You do need to have to maven set up correctly

- MuleRuntime: Download the Mule Standalone Runtime for Claude to use.

What can Claude Code do?

- As I said before Claude Code can do basically all tasks that are logically, architecture or development given enough information. Contradict to what people say it can only do simple tasks or generate small part of the codes,…

- Especially with Mulesoft where there are a lot of guardrails and components ready and a lot of tasks are repeatetive.

- Coding: This is obivious. But do try to chop it up and do small part one at a time.

- Architecture: I do like to do my own design and architecture, usually I will have Claude rate and critize it.

- Test: This is the most useful part in my opinion. You can let Claude use the Mule Runtime that you downloaded before and deploy app there, test and fix and bug that occures, all in one prompt. This is key as before they just generate codes now they have to make sure it run properly. If you build Mule Apps before you know testing is the most time consuming part this part alone make building your Mule APIs much faster and with a much hihger quality.

- Debug: This is also huge as above, with hard bug you can have give Claude session access and pull the log directly from your Cloudhub or from your ELK. Or even go as far to have a agent monitor your log 24/7 report to you through slack and fix it if necessary. A lot of company is already doing it.

- All these tasks can be run through your phone, although you might have to take a slightly bigger risk to give this many access to your Claude.

- Safety: the safest way I think is to never give production access to Claude, this way you can code as you want without worrying too much, all action in production will be done by you.

Some Mulesoft Terms that you need to understand AI output and build an integration (can skip if you are experienced with Mulesoft):

- Anypoint Studio: This is the main tool that will help you build an APIs and test them
- Arguments, properties, yaml and env: as saying above never give Claude access to your production as a safety measure. Although I would say your production data usually already in some cloud server that own by other companies.
- .m2: this is where you configure access to your dependencies, repositories,…
- Postman: This is where you store and test your endpoints, useful if you have it to test with stakeholders and users.
- Mule Runtime: is the engine that run Mule APIs, when using Studio you have the Mule Runtime Embedded underneath. But your Agents or AI can't use it so you will need local standalone runtime copies for testing
- Runtime managers: Where you deploy and monitor your APIs, you can give Claude session or full access to your log and have it monitor or find bugs etc.
- GIT – How to safeguard your API while vibecoding
    - GIT repositories serve as your back-ups or save files for your APIs. If you make a mistake and broke your API you can roll back to an older version and try again.
    - This save mechanism is quite advance, you can go back to a certain save point, create a new branch and work on the change without affecting your current code and merge it back later or pick certain point and merge it into your working branch,…
    - This will be the basis for autodeployment later, which is necessary for scaling pass 100 APIs
- **Understanding principals of GIT:**
    - Basically when you do a git init inside your project you have a local git repository, this git repository you can store anywhere you want like GitHub, BitBucket or Azure by connect the remote of your local repository to the online URL.
    - But to commit (save) your code and push it online you need to authorize whatever tool you are using you can ask Claude how to. Recommend for your terminal is using protocol like SSH for more security.
    - After you have your credential set up Claude can commit and push the code for you. Although I prefer to commit and push myself for extra security when working on client's codes. Or even set up claude own credentials with less permission.
    - You can also use tool like GitHub Desktop, Source Tree or Anypoint Studio built-in tool to connect with your repository.
    - .gitignore is the file where you add files that you never want to commit to your repo (use claude to set it up for you)

    - Make sure you encrypt and or add your passwords, secrets, api keys to gitignore so you don't commit plaintext API Key to your repository.

    - Most commands or usages can be used and explain by Claude.

At this stage I will have 2 brand only master for production or last working stage and dev for adding features. If you comfortable with it just 1 branch is also fine. Since you are the only one working on the API this keep the overhead to a minimum. Once you passed 10 APIs or integrations you can start working on branching strategy and CI/CD

Workspace settings for AI

- This section is useful to test out because it will be the set up of your agents. You understand how thing works underneath and the risks of your agents.

- Start Claude on the work space level with all your API inside one folder. Don't try to separate APIs to different integration. AI is surprisingly bad at switching folder or looking into different folder at the moment. So other token you save from having smaller workspace is not worth the headache.

- You want to have all your APIs and even external system like Salesforce Org inside also. This way you don't have to point to different folder and AI have the overview on all your System. This is also the basic of agents later on.

- Anypoint CLI

    - You can set up so your Claude can pull logs from CloudHub by themselves no more search the log in find the error and paste in the chat. You can set up so that AI can only read log and can't make changes to your APIs.

    - You will need to set up Connected App with right permission.

    - On depends on how risk tolerate your company is you can add permission for deployment etc to let Agent deploy APIs for you (same action as you manually upload jar file). Although it's better to review agent's code and go through the DevOps pipeline first before deploying changes.

- Local testing with AI

    - Put Mule Runtime Standalone in your workspace folder

    - You will need to install the correct Java

    - Your maven settings should be correct at the first few build to get all the packages needed for building the Mule app since some of the packages needed licenses or is private. After that you will only need to update the dependencies if needed. Another option is download and install them locally

- After that Claude will config your Mule Runtime for you, copy the jar file there and start the runtime. Log files will be generate in the runtime folder so Claude will read it and fix whatever bug appear.
- This is also key to have your agent develop bug free Mule codes.

How to design cheap and reliable integration solutions?

For the first 10 APIs it can be very simple. Only 3 points is important

Determine how you will start your API

- Pulling data on a fix schedule, don't need real time updates → then you can use a scheduler
- Someone need to send you data, trigger a flow or get data on demand → you need to create a http listener aka endpoint for them. This involves building a raml
- Someone will send you data automatically in the back ground → you will need to receive your data using a queue. This is usually called async pattern, event pattern, message pattern,… This should be avoid at this stage.

Determine how you will get the data

- Do the client send you data or do you have to get the data somewhere else?
- How do you get the data if it's from another system? Read files, query databases, response from another API.

Determine how you will put date into other system

This is where it get most complicated since most of the write operation is slow.

- Is it an insert or update of data? Tip: you can sometimes do both with an upsert meaning if not exist in the system insert otherwise update.
- Do you need to keep the order of the updates? Usually make sure it's synchronous and throw some kind of errors to the clients is good enough.
- How much volume is the updates? Then you will need some kind of async processing.
- A write file is architecture simplier in that it most of the time an insert of new file. So you can simply retry fail operations, usually you need to make sure it's async
- It's rare to need to keep order of a high volume of updates, if that is the case then we need more advance design which will be covered in the next chapters.

How to build your first API?

Connection

Determine how you will connect to other systems and collect credentials. This can be done very early on since it's usually take a long time. If you can look around of the systems it can be even better.

Naming

One of the thing that a lot of people ignore but quite important to scale your platform later with agent is naming. You don't have to over complicated things but make your name as informative as possible. For example: p-sales-fulfilment-acme (process api of the sales fulfilment process of the acme department) or s-ecom-salesforce (system api of the salesforce system for e-commerce). You don't have to follow any strict rule but funny enough the more readability you have the less agent or AI will hallucinate later on.

Basisc Data Transformation with Dataweave

- It's the transformation tool of Mulesoft, you can basically build different payload quickly and simple. The rules are pretty strict and straight forward so AI can generate 99% of dataweave correctly.
- You can do stuff like: changing structure, format numbers, date, transform json to xml etc, read and write files,…
- There are a lot of prebuilt functions that you can use to make your dataweave more powerful.
- Some of the most important functions at early stage are `map` and `filter`
- `map` does the **same transform/process** on each item in a collection in **parallel** this is a key point to understand. You can then use `if` `else` to conditionally mapped fields. This combo is very powerful for 80% of data transformation.
- Dataweave is an expression based language so you can't change a variable you can only set it once. So if you want to use a `for` loop you have to use `foreach` `component`
- `filter` can be use to quickly removed item from collection base on condition statements.
- You can create your own function. A lot of time you will see a lot of repeated function, mapping, these can be then moved to an Ultility Dataweave that you can then reuse across multiple Dataweaves. This is not crucial as we have AI to code for us but can be a nice to have clean up.
- Keep your DataWeave simple and easy to read while trying to do as much as you can with it. This will keep your flow simple as well.

Flow Component Basic

- Choice, Flow Reference, For Each, Scatter-Gather, HTTP Request are your bread and butter that you will use in almost every projects
- `flow reference component`
    - Call another flow or subflow and execute them

    - Can be use inside of other scope components like foreach or scatter-gather or inside dataweave

- `foreach component`
    - Equivalent of a for loop, process each element inside your collection one by one.
    - Your payload will stay the same, you will have to use variables to aggreate your results.
    - Low overhead, simple routing, easy to use. Small collection with no outbound calls (or fire and forget calls more on this later). Easy to aggregate results.
- `scatter-gather component`
    - Run the same message in multiple different routes parrallel then aggregate the results
    - Useful if you want to call or send your payload to multiple APIs.
    - Be default if one route fail all the process fail. Recommend way is to at a try catch inside each route and handle the error separately.
    - You can't modify the var directly inside the route but you get a copy in every routes and can modify that copy.
- `choice component`
    - Equivalents to if else statement, execute from top to bottom, the first success route end the scope
- `http request component`
    - Straight forward basically you setting up your Postman call inside Mule
    - The go to if you want to call an other systems. Most of the time prefered over prebuilt-connector.
- Once you master these you can build 80% of integration. Afterward we can start working on more complex integration with asynchronous pattern.

Integration vs REST API: What is RAML

- Use to define your endpoints, uriParams, queryParams, dataType,…
- With RAML you can enforce and have your json structure validate against a spec
- You can define it inside Design Center and have a live mocking API from there or use API Console to test your API
- It's useful when you want your APIs to be used be AI or agents.

- If your API is use only internally especially when you are SOLO you might not even need spec, that's way it's much easier to add changes etc in early stages.

External Systems Basic:

REST API

- At this level focus on all the Rest API that you can connect. It's simple and get you're your first quick wins. More and more system offer built-in rest APIs so you don't really have to build complicated connector.

- Tips using a transform message to prepare your request first. For example: Variable - httpPath, Variable - httpQueryParams, Variable - httpHeaders, Variable - httpBody,…It's make it easy to read and maintenance.

- Put your body last in the transform message since that will be the most changes happen and it will be the default opened when you open the transformesage.

- Path and method can be externalised into properties files.

Salesforce Basic:

- How to connect to Salesforce:
    - For Mulesoft there is no user interaction and usually is server to server so the best fit is External Client Apps with OAuth Client Credential
    - Step 1: Create an External Client Apps and fill in the information
    - Step 2: Enable Oauth
    - Step 3: App Settings
        - Callback URL: (the url that the token will be sent) any urls
        - Oauth Scopes: these decide what your API can access
        - Ignore other 2 check boxes
    - Step 4: Flow Enablement: Client Credential
    - Step 5: Security: Enable "Require Secret for Web Server Flow" and "Require Secret for Refresh Token Flow", disable "Require proof key for code exchange (pkce)"
    - Step 6: Click on Create
    - Step 7: Go to Policies Tab: "Enable Client Credential Flow" and set the user you want to run as, set your "Refresh token policy" and "IP relaxation"

    - Step 8: Go back to App Settings and get the Consumer Key and Secret and Verify your request. Save the credentials into your Mule APP
    - Note: You will need all the External Client Apps Permission add to your user to get the Consumer Key and Secret

- How to pull data from Salesforce:
    - Query is recommend way to pull data from salesforce since you can add multiple filter to your query.
    - Salesforce do not let you pull all fields of an object so you have to actually know all the fields you need.
    - Write all fields in a column for easy reading
    - Custom field have __c at the end
    - It's better to write soql in sf the component to prevent injection instead of set a string as variable
    - When using condition make sure you handling the empty result case properly because Salesforce will only return an empty array without any error message.
- How to write to Salesforce basic:
    - Insert when you want to strictly cancel the insert process when the record already exist.
    - Update is used when you want to strictly update records and cancel if the record doesn't exist in Salesforce
    - Upsert is the more common way to insert or update records, when the record already exist it will update your record, if not it will insert. This way you reduce the overhead of handling different cases.
    - Payload should be java since it preserve the datatype. And it has to be an array (list of map)
    - Name is usually required for a new record unless it is an auto number.
    - When upsert failed you won't get any error so make sure you check the response properly and raise error if necessary

Chapter 3

From 10 to 50 — Building the Foundation and Workflows

What are Integration Patterns and when to use them?

Integration patterns are useful when start run into integration that have high volumne or long processing time. These pattern then will help you understand what you are building and why.

Synchronous pattern

- The default
- Basically your make an API that listen to request, return a response or and error.
- Simple to build, set up a http listener, wait for a request then return a response. On error throw an error and let the client handle it.
- On the synchronous pattern, you need understand that the client will wait for a response so there will be a time limit to process your request. Which becomes problem if there are too many requests, the request is too big or the processing is complex. This usually can't be handle by the client alone so you need to implement another pattern for it.

Scheduler pattern

- The scheduler pattern is a simple way to handle the problem from synchronous pattern, you run your flow only in certain period, there is not client so you don't need to reponse and free to implement async pattern in your code. If you failed you can pull the same data and do it again.
- How to do it: Simply replace your trigger component with the scheduler component is usually enough. Make sure to have proper watermarking process.
- It's simple and predictable and you have the space to build more asynchronous flow without worry about responding to the requester
- The trade off is you lost the real-time processing and you have to maintain the watermarking to make sure you not processing again the same data. And if the period is too short or the amount of data is huge you have the same problem of processing in the time limit.

Event pattern

- Whenever there is an change event you get a message. You use a queue to collect these events. then process them.
- The queue removed all the contraint of waiting for a response. You take a message and process as long as you need. Most of the time it is fast enough, since you are free to use async components, that it's very close to real time while decoupling your systems.
- With it come a problem of when the message fails, then your source system will never know cause you never send a response back.
- For this problem you can use something call a Dead Letter Queue. Basically when you fail to process a message you put that message it a DLQ for reprocessing or review.
- For even deeper async or fire and forget implementation you might need to pull the result to check.
- How to implement it:
    - Most straight forward way is using Anypoint MQ everything is already built in
    - You can also use any other message queue of your choice
    - From publisher side you can't expect a response so there won't be stuff like external id or result,… You will need to pull later for those
    - From listener it will be mostly the same, you can set up to pull huge amount of message or one at a time.
    - Error handling will be a bit tricky as mention above.
- You can also implement Asynchronous Pattern without any queuing which mostly not recommend unless you use scheduler pattern where the change will be update in the next run.

Asynchronous Components

We will look at a few Asynchronous Components that can be use to increase performance of your API.

`async component`

- Basically you run a sub flow or part of your flow in a separate thread. Think of it as Mule create a child app to run your process. The main flow will continue and forget this part regardless of your `async component` and your main flow won't get any response back from this component. But you can persist it somewhere by writing it somewhere
- You can add a `try scope` to handle error by write an error message the log or DLQ
- Can be useful to quickly spin up a thread to run your component. For example upload files you need to upload a file or write to Databases

`parralel foreach component`

- This is the foreach component but process all elements in parallel
- Why do you need it?
    - large collection that doesn't need the processed elements in the end to be in the same order or share state
    - or process that doesn't need to stop if 1 element failed
- Same as foreach but results can be accessed in the payload.
- Default is one element fail the whole flow fail, use `try scope` with `error continue` to partial processing.

`batch component`

- It's a extremely granular parralel processing component. It process big payload that can't be load into memory.
- It process each record separately in batch step.
- You can use filters to process each set separately.
- You can aggregate the results in each step to do bulk actions like collect 200 processed records then insert into Salesforce.
- Implement each step as you processing 1 item.
- By default if one record fail, the whole thing fail. You need to set `maxFailedRecords` to acceptable number.
- Make sure all your errors are throw and be careful with error continue inside batch step.

Memory management

- A common architecture level error is you load a big payload into memory instead of streaming you risk run into this error java.lang.OutOfMemoryError. For example: using for each or for each parallel on large payload.
- But streaming is on by default in most connectors. There are pagination and batching from upstream that make the payload smaller.
- The math is `heap pressure = (size of payload on memory) × (number of copies/in-flight instances) × (concurrency)`
- Things that could cause problem:

- Parallel For Each with large payload it will load all into memory, with large collections it will spin up a lot of concurent processes by default (maxConcurrency is unlimited).
- DB selects without streaming="true" (most common cause of OOM I see)
- DataWeave with sizeOf(), orderBy(), groupBy() on streamed input (silently materializes)
- Logger printing full payload
- Cache scope without size/TTL bounds
- Accumulating multiple large variables across flows: Scatter-Gather with large per-route responses, aggregating into a flow variable inside a loop
- Too many concurrent requests
- Misconfigured and load too much data in batch job record blocks
- Third-party connectors that buffer internally
- Processing a 2GB monolith instead of paginating, batching, or filtering at source

Why you shouldn't use CDM or Domain Bounded Data Model below 50?

- A lot of team has CDM as their default in first iteration, or as selling points. On paper it sounds very nice, but in reality there are alwas trade-offs.
- It's hard maintain, create frictions in communication which increase chance of mistakes. Only beneficial if you have multi sources and targets that have overlapped data models. Even then Domain Bounded is more recommended
- Especially it should not be in your first iteration. Because things evolving really fast, the moment you get the CDM to a decent level, things already changing and it stopped being so useful (negative ROI)
- Me personally I would only introduce CDM and even RAML when I need to especially serve external clients. If API only run internally I would just use the data model of source or targets system. Because you are SOLO it will be to your advantages since stakeholders will only need to know the mapping between sources and targets.
- Most project, CDMs are just built on top of the System API which just increase the mapping that you have to maintain to 1 without any real usages.
- Yes it can be easier to read for external developers but most systems have hard to read fields anyway. Nowadays most codes will be read by AI anyway.
- With AI reusabilities is in my opinion uncessary. It takes a few seconds for AI to read and write. By the time you implement, use, maintain your libraries, you could have build and deploy

multiple APIs. This can be done nice to have as a clean up, late stage efficiencies when you have your platform at a stable stage.

It's time for RAML Datatypes and Resource Types

- Datatypes or Resourcestypes is then when you genuingly repeated RAML even then with AI it take a few second to copy paste it and sometimes it easier for AI to manage one big RAML file than multiple small RAMLs
- Similar to RAML I think unless you serve your RAML to other people no need to use Datatypes or Resourcestype.
- The logic is if you use CDM then maybe Datatypes then posible resources type when calls are also repeated.
- I remember in one of my project there was 2 devs when I joined, both are not Mulesoft Specialist, one joined after his PHD one was the internal very Senior Java Dev. The team hasn't ship an interation yet but they have built a massive salesforce API and process API with datatypes and resourcetypes waterfall style.
- Structurely it look clean but there was bugs coming everywhere since the build was never end to end tested, change requests are super tedious to do since you need to update multiple files.
- Slowly pre prod codes built up to massive amount that's it's too scary to deploy.
- Intuitively I know it was problematic but I was young and thinking that the senior should know better than me. Obivious it comes down to politic and I'm not quipped enough at that time to fix everything. That takes us to the next part.

Should you pack all your integration into 1 API?

- It's kind of ok now when you are a solo dev using AI.
- Technically your API are kinda more vulnerable since it is a single point of failure which can be mitigate somewhat with the increasing of replicas.
- With multiple devs you will want to avoid more than 1 dev working on the same API at the same time as much as possible. So packing all your integration into a few APIs should also be avoid.

CI/CD Auto deployment

- After you have your Git Repository set-up a CI/CD pipeline can be used to deploy your application automatically whenever you commit or merge a new change. In the case of GitHub it will be GitHub Action.
- This become necessary I would say around 10 production APIs, me personally I'd like to set it up earlier. As you work solo you don't have to take the jar files and upload it manually anymore and you can track which changes you have in your deployed APIs.

- Early to prod is the key to success. Do it small and incremental so you don't risk breaking anything too big at once.

- A lot of teams being too careful about prod deployment, which in turn create more bugs and problems, as fear breed problems and code which are not battle tested often enough will be fragile.

- You can't prevent prod problems. we saw even big tech and billion dollar companies have bugs, got hacked, leaked,... even with the best devs the most comprehensive deployment processes.

- The only way to reduce prod problem is to go through a lot prod deployments.

- Make the deployement process lean and fast enough that you can put a hotfix up quickly no matter what happens. And since you don't ship big changes bugs and problems are easier to find and fix. As the result you reduce the fear of prod breaking and in turn reduce prod itself breaking.

Branching strategy for deployment

- Ideally in my opinion you need 2 enviroments for Mulesoft: a Production Enviroment and a non-Production Enviroment for testing end to end.

- Some clients are more lax where you can do some testing on Production. Still you should be careful not to do mass actions like delete or update records.

- There is a new trend among indie hacker community to develope everything straight on Production. Which massive reduce friction from shipping. This can be strategic move if you are a building the platform solo from scratch and if the client are open to it. You will learn how to do proper coding on first iteration instead of write bad code and then fix later when ship to production. And also a lot of time set up correct testing enviroments is a huge task of its own that creates problem of its own. As long as you don't mass delete records and follow proper guidelines you should be safe.

- Some clients are more strict where you need an extra dev enviroment for developments since test or UAT will have data that other dev team need for their own test.

- I have been in project where there are multiple enviroments there are testers etc. But the codes are built up sprint after sprint into a massive blob cause people have anxiety when deploying something to production, and it would always be postponed. Then APIs were never shipped or if it was shipped there so many bugs at once that team scrambled to fix which in turn create even more fear for next deployment.

- Shipping small pieces quickly to production have the effect of reduce the size of the roll out, the change of breaking something big is reduced.

- While if you adding 1000 commits together in one version after every 2 months and deploy that into production, no matter how much you test you will get more unexpected bugs and errors, chance of breaking is higher and take longer to fix and prod will be down for longer time.

- The things is Mulesoft already has many guard built-in rail to help you to ship faster which is completely useless if you then fall into waterfall trap, premature optimization, premature preventing bugs, don't develop together with design, don't ship until the whole feature completed designed,…

- Ship fast also help you to adapt faster to different situations since you will be working with many different teams and systems. Which is the main struggle of integration dev.

- What I'm saying don't be too closed minded, there are always traded off in software development. And there are situation when it's better to ship quickly to production and there are situation where test enviroment is a must. Nothing is always applied for every situation.

How to 10x your test with AI?

- The overhead of testing was really high in Mulesoft. Local debugging is slow cause of Anypoint Studio. Deploying multiple APIs local is too tedious and time consuming.

- With AI Tool like Claude Code now you can deploy multiple APIs locally without Anypoint Studio, you can edit the codes, redeploy and test it with a few prompt.

- Testing End to End is still somewhat slow since you need to deploy to Cloudhub have test data etc ready. But you can debug quicker cause AI can read the log 100 times faster than you do.

- This is why in some case like new system, no critical, important data it's beneficial to test live in Production if possible, of course with heavy monitoring. They risk are really low since we already have our local end to end test set up almost as a real test enviroment.

- Or if not pull real data from production for testing without writing.

- The more context LLM has the faster it is to ship and fix bugs.

- Now you can put your resource into shipping and improving your architecture and user stories.

Should you pack multiple integrations into 1 API or follow what Mulesoft recommend to separate API into business domain?

- Before AI, this is a gray area. Yes the cost of multiple APIs are high but it makes your platform much more stable and easy to manage since each process is isolated from each other. Change on invoicing won't break employee integration

- With AI it is much easier to test and debug which reduce the risk of breaking and less developers working on the same API reduce the risk of errors in one integration break another integration.

- So it's usually fine to separate codes into different files and share 1 connection config. Although if the process is really critical you want to have a separate API handle it. And not just purely logical or system based.

Should we have mapping in system API?

- This is usually not a clean architecture, especially if it an API that you pull a lot of data from. Now everytime you need more or different data you need to make change to the system API. Which you run into the risk of breaking other integration. With AI it's easier to manage change so in my opinion it is acceptable to do. But still have a separation if possible.

- Now if we have both multiple endpoints and mapping into one API you run into the risk of having a central point of failure. This a good time to consider adding replicas or workers if the API is critical.

External Systems Intemediate

Salesforce Intermediate

- Bulk API:
    - It uses to async upload a large amount of data more than 2000 records into Salesforce.
    - You send a bulk job request, you get a path to then upload a csv that contains all the records that you want to insert, update etc,
    - then you update the job state to UploadComplete.
    - Afterwards you can track that job state by querying the result
    - When the job complete you get a csv with Salesforce ID and if the records is successful insert or upsert
    - Similarly you can also send a bulk query job for large set and collect CSVs.
- On-modified Listener:
    - If you don't need to listen to the change in real-time and the volume of changes in a period is not large
    - Basically you pull Salesforce for new changes similar to scheduler pattern.
    - This can cost more API Request than an Event Pattern.
- Common problems when connecting with Salesforce:
    - Permissions is the more common than you think that Mulesoft Devs usually miss: Salesforce Admin added a new field or deploy the fields and forgot to deploy or add the permission and the field level security for the integration users.
    - Field label and API Name: A lot of time org have multiple fields with similar labels, so misscommunication can occure if Salesforce Admin doesn't have experience with integration. Always double check.

SAP S/4HANA

- Most of the you ended up just connect to their standard REST/SOAP api.
- Customization is quite expensive in S/4HANA for example if you want to add a field to the standard API you need to extend to a second custom API call. Which is good in the way that you don't need to gather the mapping but you can just use the standard ones most of the time and cover most of your integration
- All the documentation are public and searchable so easier to work with compare to SAP ECC that even documentation is inside a paywall.
- So with AI tool you can just give the service name (Endpoint Name) and it can built the whole system call and mapping for you.

Netsuite

- Similar to S4 you have an extensive documentation so you can usually let AI do the mapping and building for you
- You might want to use HTTP Request instead of the Connector
- The timeout max is 60s for the Connector so you might need to configure your pageSize setting so your request failed (Remotely Closed) before getting the response

Chapter 4

From 50 to 100 — How to Scale Your Platform

What to focus on after 50 APIs

- After you build up to 50 APIs, you will need to spend more time improve the architecture. Some will say it's too late but in my opinion this is more efficient.
- You will see pattern that emerge, recurring problems, unreliable systems,…Now you can optimize what is actually needed. I hate to repeat this but premature optimization is the root of all evil. See where you can add async, switch to a batch or bulk pattern…
- For the codes make sure your codes are organized and easy to read. Don't use AI for this, actually going through and review and try to read it as a junior dev. If it's too complicated to read make it easier.
- Contrast to mainstream narative where you need clean codes, modularization,…, as a solo engineer, readabilty is your best friend.
- With AI you don't need to worry about manually adding changes, typos,… so duplicates codes are fine. Too much modularization makes it hard to understand and review changes.

Scale with Common Data Model (CDM) or Domain Bounded Data Model?

- The point of the CDM is not to reduce the amount of codes but to reduce the amount of mapping that need to maintain. If adding CDM make it's harder to maintain your code, you shouldn't add it.
- At this point you should have a good idea of mapping that can be consolidate into a CDM
- Compile a list of mapping for source and target systems. If there are multiple sources and multiple targets that share the same field mapping.
- Most people mistake about about CDM and only focus on source, and they ended up just abstract the system layer which is fine you have nicer field names but not worth the overhead.
- After you have that list of mapping, create a CDM easy to understand fields name. For example **WERKS** in SAP to **plant**. You can rebuild the whole structure how you see fit. This sometimes called **Bounded Context Data Model**.
- After you implement a few Data Models and notice that even these are repeating. You can abstract even further to create a CDM. A true CDM is then a model that covers all different version of that object in different system.

- You can also go the other way and just use one of the source or target system model as the CDM
- This helps with communication with other stakeholders and building later mapping and in turn integration faster
- The problem is most consultancies and Architects will spend a lot of resources building these models right from the start, and massively increase overhead on the platform before even ship any integration. Which as explain previously not the best way to build Mulesoft APIs
- If you really want to build it early build it parralell with your APIs. That help you build faster and tested your CDM at the same time which makes your CDM even better.
- Avoid Waterfall trap where you build your CDMs for months and have like 50% of it unrelated to your integration while missing the other 50%.

On-Premises deployment

- Mean that you will deploy your API on your local machine instead of cloudhub or your private cloud.
- It's used to be too costly to maintain so almost you will never see any teams attempt it. With AI this is much more attractive.
- You can deploye with runtime fabric, which is an Anypoint Platform wrapper. You APIs can be fully managed with Anypoint Platform, you can use replicas, internal load balancer, SSLm, Auto-restart on crash, Auto discovery,... this is useful if client need certain data to stay in certain region which often happen with personal data in Europe. But the cost save is not much.
- Without runtime fabric you will have to install Mule Standalone Runtime on your machine and configure everything yourself. You can use Mule Agent to connect Anypoint Platform in a hybrid option. These are the real budget option if you want to save on cost.Around 50 APIs mark you want to look into this. Deploy APIs with high load can save you around 20-30k usd per vcore you migrate to hybrid.
- This is good for APIs that are small, low-medium load, latency-sensitive or internal low-priority back-office.
- Services like Hetzner offer really cheap server.

How to onboard more MuleSoft developers?

- A big trap that a lot of manager got into is hiring someone who learned too much best practices, good at coding interview,… These developers tend to have a set of problem that detrimental to effectively shipping APIs with AI that I mentioned above:
    - Too rigid, too careful leading to premature optimization, missing deadline, unable to adapt to different systems and stakeholders. Which is bad for integration team.

    - They usually focus too much on technical stuff, need to improve their skills and their CV so they makes things more complicated, overengineering.

    - Because of these they are usually try to bend business stakeholders and users to them. Which run into the classic problem of looking at the techs first and then try to find the user stories.

- We want people who can understand the users's problem and build solution simple and fast even if it's not the most clean engineered feature. You should be able to throw a problem at them and get the result back quickly.

- Always look for hybrid Developer and Architect, better yet get a Developer/Architect/Business Analyst hybrid on your team someone who not only can solve integration problem and also know how to work with stakeholders and users and find the right problem to solve

- If you are not a Mulesoft Specialist, you can go through this book and get a bit hand on then it will be easier to see who is a good developer who isn't. If they checked the boxes for philosophy and mindsets. Give them a trial run for 2 weeks or a month and decide if they can deliver for you.

- I also offer Mulesoft Consultant service, get in touch with me for my availability.

Documentation — The key to maintaining your power as a MuleSoft Platform Owner

- Documentation is always the biggest hurddle for MuleSoft Devs. So how would you as a Solo Dev handle Documentation? Yes when you use AI this will be much easier but then also where you will possible spend the most amount of time.

- A few key principals that you should keep in mind:

    - **Focus on the reader**: if you write a feature or an user story, focus on the stakeholders, if you write an API Documentation focus on the other devs. Don't be too rigid that you only write feature a certain way or user story a certain way and your users should adapt to it. If your stakeholders are not very technical useless technical terms, explain things more simple,…If your devs are juniors, write more details, give more intructions.

    - **Don't need to be perfect**: Most devs I see always stress when writing a document because they worry that their seniors or fellow devs will judge them on it. As the result all the documentation fall on the senior devs, architects or managers. They can write the most complicated codes but struggling to explain it in the documents. Most readers will only need to read a certain part of the doc and find what they need, they don't care if the doc look nice, have a good structure, etc…

    - **Keep in as simple as possible:** add and update as you needed when you collect more reader feedbacks. Have AI generate the complex version for you then edited it down to the key points.

External Systems Advanced

Salesforce Advanced

OAuth JWT

- More complex to set up
- Private key never travel so extra secure
- certificate rotation for extra security

Triggers + Platform Events:

- Standard way to receive large amount of data or receive data in real time.
- You don’t have to worry about keeping the watermarking.
- You need to configure Salesforce more.
- You can only fields that are needed to the platform event and save on API requrest, latency and memory.

External Object

- You can show data in Salesforce without saving the data inside your org
- You have to make your RAML OData compliant by using OData RAML definition
- Then you can have your data show in salesforce without actually saved it inside of Salesforce.

Local On-Premise Database or File Server

- Sometimes you will need to connect to a local database or file server for sensitive information.
- Check if you need a VPN connection. This mean you will need to have a private space on your Cloudhub that connect to this VPN.
- Otherwise the standard is the API IP is in the whitelist and the server open the right port (TCP 1433 for SQL, TCP 445 for SMB Window Fileshare, TCP 21 for FTP, TCP 22 SFTP.

SAP ECC

- On the point of On-Premise Server we have to mention SAP ECC.
- It’s a legacy ERP System that usually deployed On-Premise.
- Closed source so need some experience

- ABAP: Is the language of SAP Business Functions, think of it as APEX. Most Business Functions that you interact with in SAP is ABAP underneath.
- RFC - Remote Function Call:
    - Is the Transport Protocol of SAP ECC. It's like HTTP but specialized for SAP functions.
    - You can enable your ABAP to allow this then call it remotely from your Mule API.
    - Or you can called the standard RFC Functions
- BAPI - Business Application Programming Interface
    - SAP offer a more standardized way to communicate with RFC Functions called BAPI
    - These follow a stricter pattern of reponse, commit and errorhandling. Think of it's as REST API.
    - There are standard BAPI and you can also write custom BAPI. Basically a custom RFC that follow BAPI Conventions.
- IDOC
    - Messaging protocol of SAP
    - When you send an IDOC there are 5 checks:
        - MuleSoft level error: similar to other systems
        - JCO and RFC level error: the JCO will check the shape of your IDOC against SAP before sending it to the queue. If any error happen before IDOC landed in SAP you get a normal error message
        - An IDOC land in SAP, you can see the shape of the IDOC and the status green or red as the status of the processing of the IDOC and the error code.
        - Sometimes even if an IDOC is green: the ABAP function that should have picked it up and process it failed. So even if you send a green IDOC you can't still failed to create or update an object.
        - From step 3, either you need to write a ABAP function to check and send the response to Mule API or doing a query to check. Since you won't get an error after IDOC is landed in the queue successfully.
    - IDOC is good for bulk and async messaging where you don't need an immediate reponse of success or failure.

Chapter 5

100+ — How to Always Keep Things Under Control

Making change fast with more than 100 APIs?

- A platform that has more than 100 APIs is certain to constant changing, new requirement, new processes, new automation,…
- One of the key things is to be able to quickly adapt to change of other systems. If you can adding changes quickly and reliably, you lesslikely to instroduce bug but also fixing the bugs and errors faster.

MUnit

- When you have more endpoints and integrations than you can remember and want to make sure that new changes doesn't break your APIs you should introduce MUnits into your. APIs
- Why not ealier, since most your code written by only you and AI the error that can be test by Munit usually already taking care of by AI. Usually with more than 2 devs you already need it.
- If you still solo at 100+ APIs, this is small safe guard but if you onboarding new devs or leaving and handover to the client it's a good delivery to add for the client.

Get the feedbacks

- Sound boring but the best way to handle change is to be proactive with them.
- As mentioned before when you actively checking on the users you get better view on the problem, more info for protential problem and you are the one who suggest the change instead of tickets coming to you when it's too late and you have to rush to implement it.

Keep your platform more stable at 100+ APIs?

- As a solo Dev actually it's much easier to manage the platform since you are the one who built everything, set everything up. You and your AI friend should have the full view on the platform. But bugs and problems will still happen.
- Have a good bugs tracking system. Often I see a lot of team found a bug and quickly rush to fix it. It's work 80% of the time but it's the bug that need monitoring is the problem. Since most of the obvious can be fix by AI quickly. A good bug tracking system solve this. Since you can't possible master all systems that you integrate with you need to collect all information you can to be able to solve this. As written below you will need to involve not just AI but multiple teams or people so context matter a lot.

- There are 4 type of difficult Bugs
    - System Quirk Bug: SAP 0 padding, Netsuite Paging Bug, AMQP AlreadyClosedException or just simply no error response.These bugs require experiences on that system, or workaround that sometimes AI doesn't have since there is no documentation on it.
    - System Actual Bugs: On-prem File System availability, File Server File Formatting,…These bugs can't actually be fixed in Mulesoft but the other systems, and can also be hard to find if the system doesn't handle it properly.
    - Architecture Bugs: Often when you take over an old implementation or you are inexperience with design the architecture. It can often be not optimal or even wrong which sometimes can be subtle until you facing bugs in production. Especially when it's your own architecture. Something like maintaining messaging order of an event architecture or commit a transaction at the wrong point when it's actually error. Can be quite easy or difficult depends on your experience. This point you somtimes need someone more experience than you to fix which can be hard when you are solo. Better to have connect to a community of architects where you can pick their brain or look up in books outside of Mulesoft. Creativity matter here.
    - Performance Bugs: Somewhat related to architecture bugs, sometimes you get a wrong estimate of change events so the integration will actually fail cause of overload. This is somewhat more obvious than architecture bugs.
- Other than difficult bugs mentioned above which can't be fully fix with just log.
- Most small bugs inside Mule can be fix by AI, let it read the error log (you can give them a c-curl so they can grep the cloudhub log for that session) or better yet deploy an Agents that read the log periodically. There is Anypoint Platform API avaiable for this.
- A good bug tracking system require a good logging system.
    - Use json logger to format your log for better reading
    - Log the position and the necessary information for debug like IDs, Names, Error message etc…
    - Good pattern is log before and after a call out, http request
    - Make sure not too log full payload unless there is no other option. In production always turn off full payload log.
    - Correlation ID will change after messages go through the queues or non Mule API so always send extra correlation ID with

Afterword and Contact

Thank you for reading this book. I'm always looking forward to hear your feedbacks.

You can contact me at **quang@nqnconsulting.com**.

If you are looking to work with me follow my email newsletters for the lastest availability updates at: https://quangnguyen.berserkermail.com/nqnconsulting