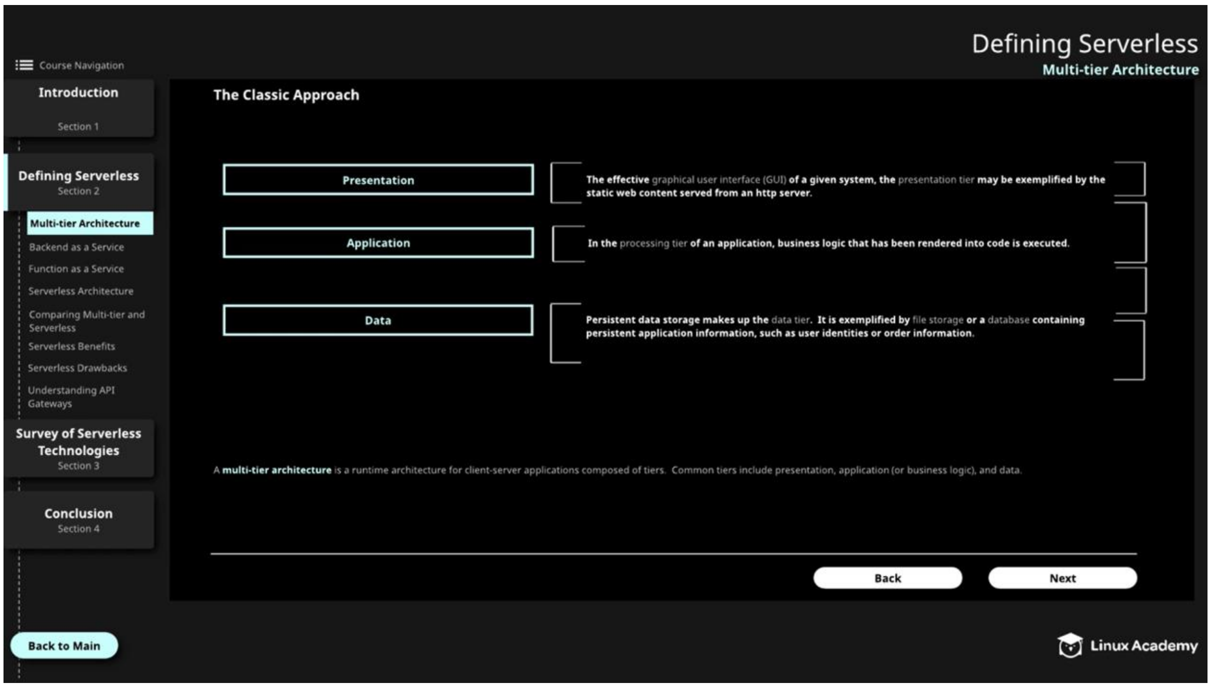
Serverless Concepts:

1. Multi-tier architecture:



Defining Serverless
Multi-tier Architecture

The Classic Approach

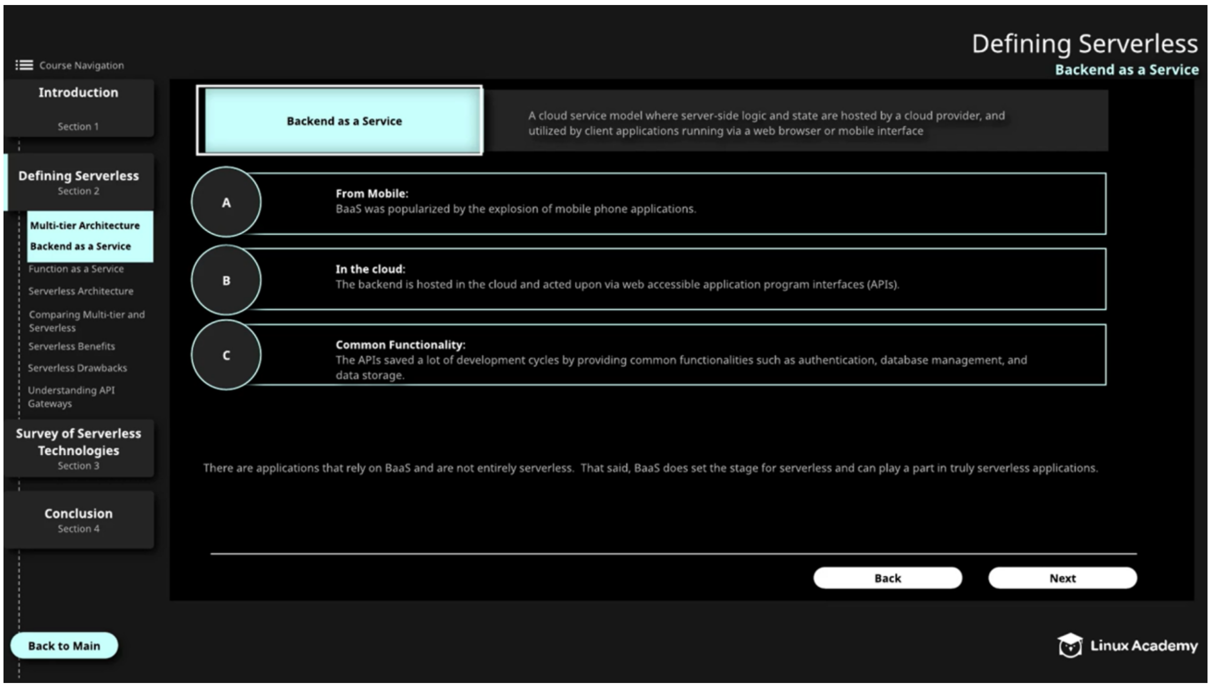
Presentation — The effective graphical user interface (GUI) of a given system, the presentation tier may be exemplified by the static web content served from an http server.

Application — In the processing tier of an application, business logic that has been rendered into code is executed.

Data — Persistent data storage makes up the data tier. It is exemplified by file storage or a database containing persistent application information, such as user identities or order information.

A **multi-tier architecture** is a runtime architecture for client-server applications composed of tiers. Common tiers include presentation, application (or business logic), and data.

2. Backend as a Service:



Defining Serverless
Backend as a Service

Backend as a Service — A cloud service model where server-side logic and state are hosted by a cloud provider, and utilized by client applications running via a web browser or mobile interface

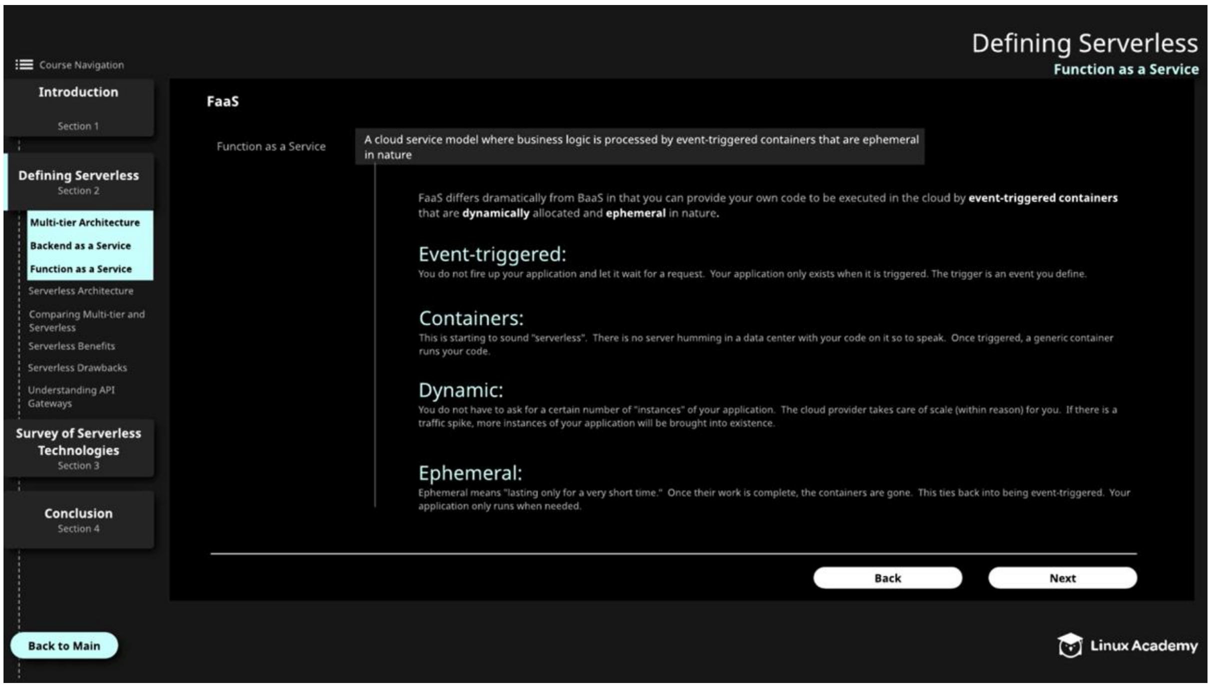
A — **From Mobile:**
BaaS was popularized by the explosion of mobile phone applications.

B — **In the cloud:**
The backend is hosted in the cloud and acted upon via web accessible application program interfaces (APIs).

C — **Common Functionality:**
The APIs saved a lot of development cycles by providing common functionalities such as authentication, database management, and data storage.

There are applications that rely on BaaS and are not entirely serverless. That said, BaaS does set the stage for serverless and can play a part in truly serverless applications.

3. Function as a Service:



Defining Serverless

Function as a Service

FaaS

Function as a Service: A cloud service model where business logic is processed by event-triggered containers that are ephemeral in nature

FaaS differs dramatically from BaaS in that you can provide your own code to be executed in the cloud by **event-triggered containers** that are **dynamically** allocated and **ephemeral** in nature.

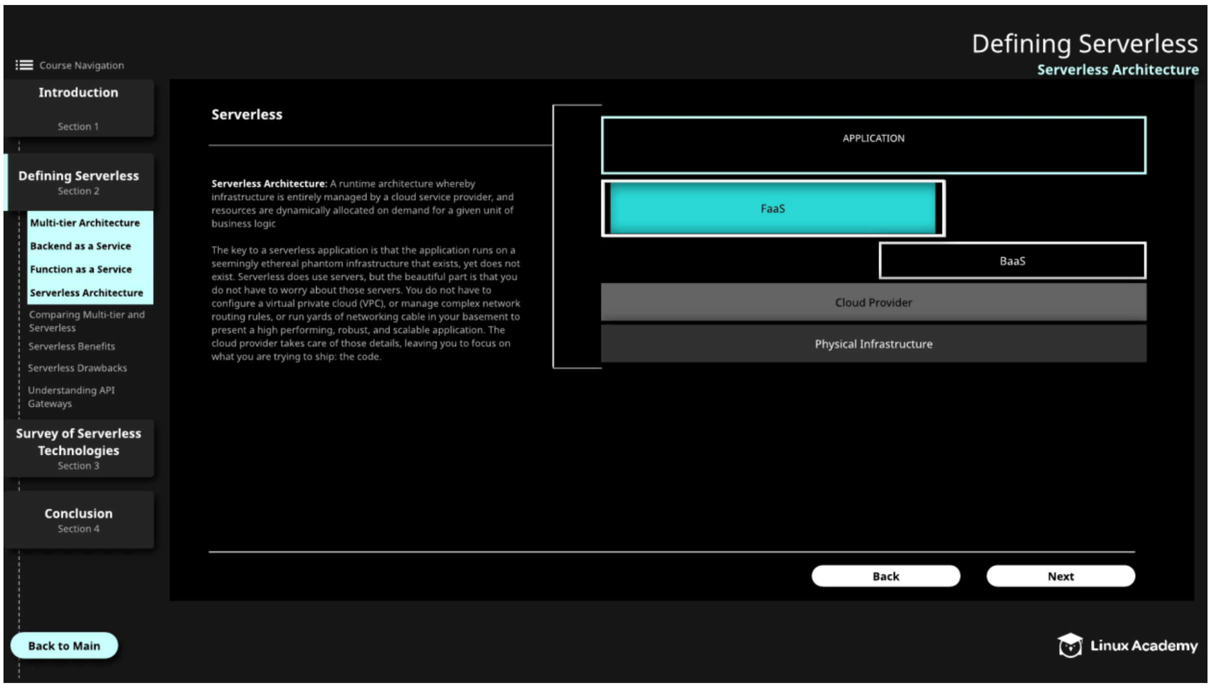
Event-triggered:
You do not fire up your application and let it wait for a request. Your application only exists when it is triggered. The trigger is an event you define.

Containers:
This is starting to sound "serverless". There is no server humming in a data center with your code on it so to speak. Once triggered, a generic container runs your code.

Dynamic:
You do not have to ask for a certain number of "instances" of your application. The cloud provider takes care of scale (within reason) for you. If there is a traffic spike, more instances of your application will be brought into existence.

Ephemeral:
Ephemeral means "lasting only for a very short time." Once their work is complete, the containers are gone. This ties back into being event-triggered. Your application only runs when needed.

4. Serverless Architecture:



Defining Serverless

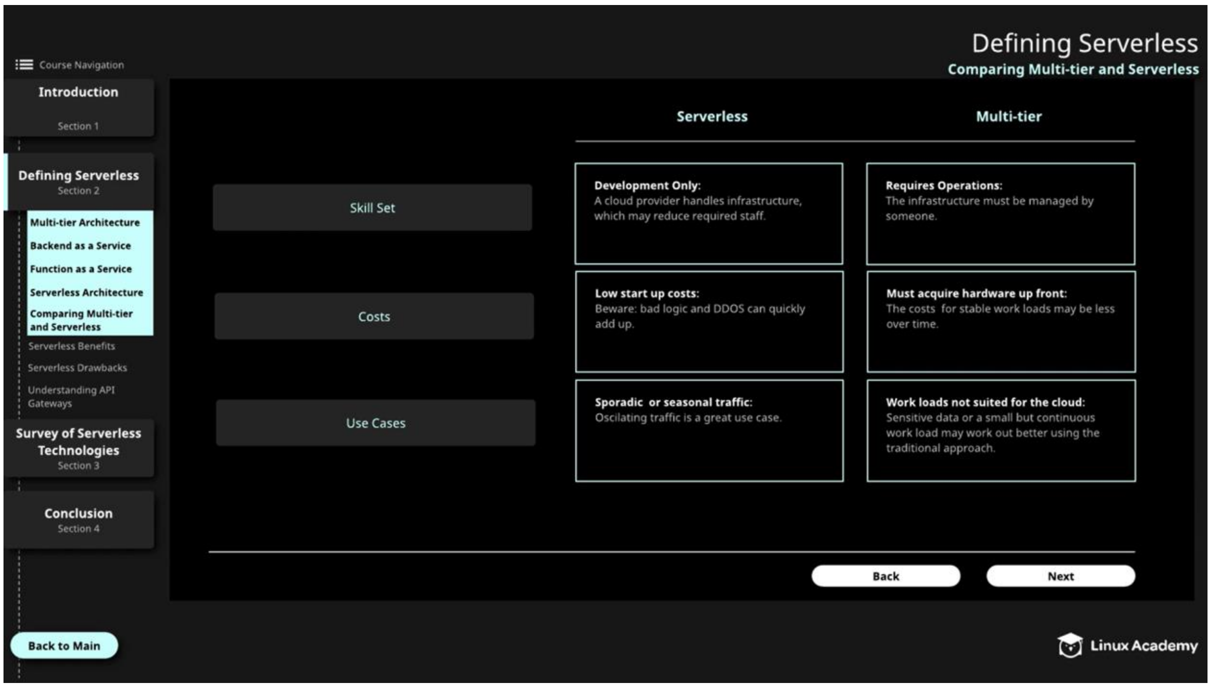
Serverless Architecture

Serverless

Serverless Architecture: A runtime architecture whereby infrastructure is entirely managed by a cloud service provider, and resources are dynamically allocated on demand for a given unit of business logic

The key to a serverless application is that the application runs on a seemingly ethereal phantom infrastructure that exists, yet does not exist. Serverless does use servers, but the beautiful part is that you do not have to worry about those servers. You do not have to configure a virtual private cloud (VPC), or manage complex network routing rules, or run yards of networking cable in your basement to present a high performing, robust, and scalable application. The cloud provider takes care of those details, leaving you to focus on what you are trying to ship: the code.

5. Comparison between Serverless and Multi-tier:

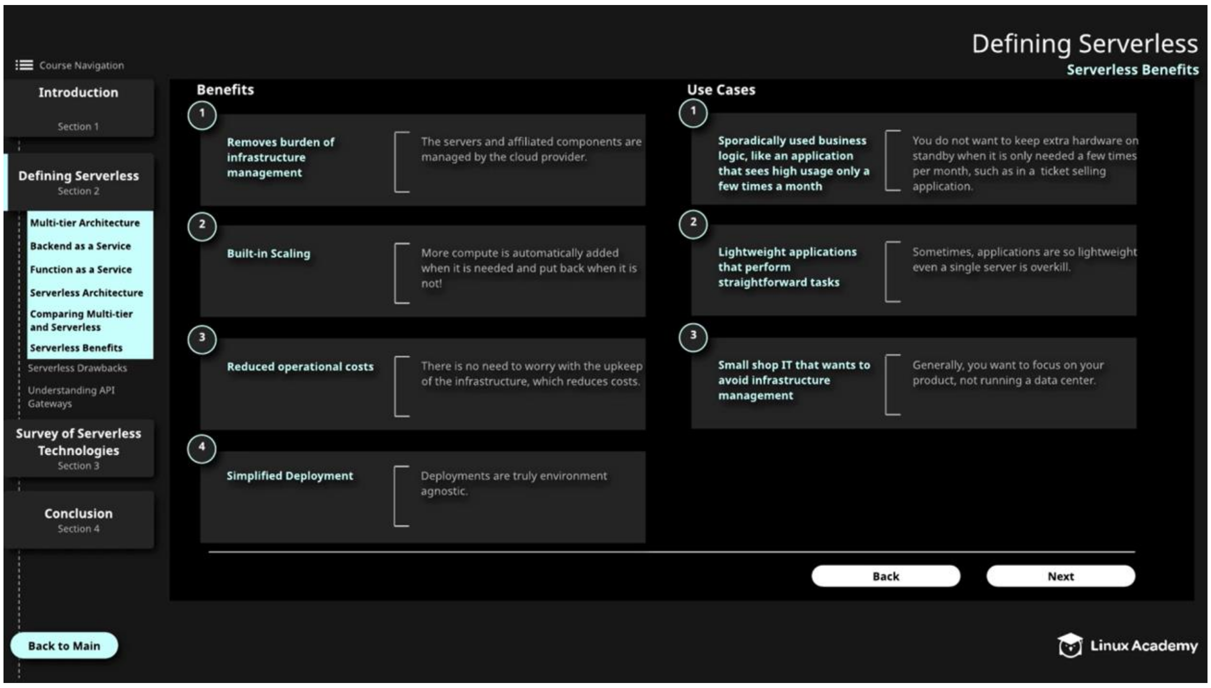


Defining Serverless

Comparing Multi-tier and Serverless

	Serverless	Multi-tier
Skill Set	**Development Only:** A cloud provider handles infrastructure, which may reduce required staff.	**Requires Operations:** The infrastructure must be managed by someone.
Costs	**Low start up costs:** Beware: bad logic and DDOS can quickly add up.	**Must acquire hardware up front:** The costs for stable work loads may be less over time.
Use Cases	**Sporadic or seasonal traffic:** Oscilating traffic is a great use case.	**Work loads not suited for the cloud:** Sensitive data or a small but continuous work load may work out better using the traditional approach.

6. Serverless benefits:



Defining Serverless

Serverless Benefits

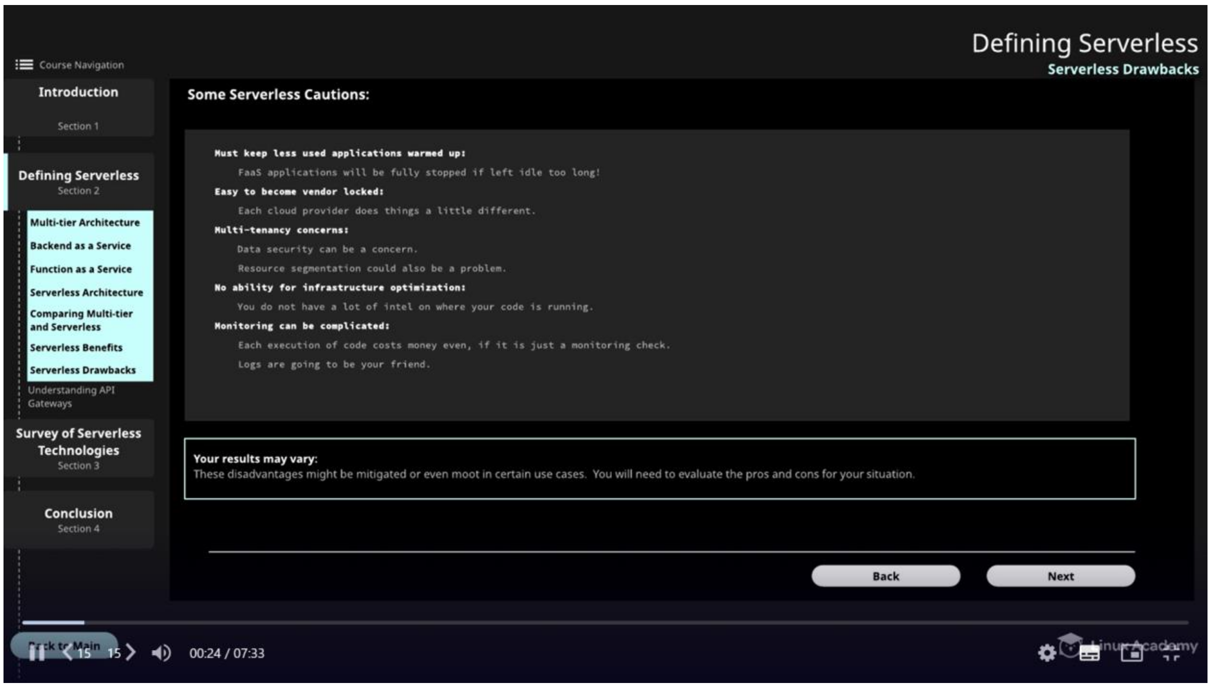
Benefits

1	**Removes burden of infrastructure management**	The servers and affiliated components are managed by the cloud provider.
2	**Built-in Scaling**	More compute is automatically added when it is needed and put back when it is not!
3	**Reduced operational costs**	There is no need to worry with the upkeep of the infrastructure, which reduces costs.
4	**Simplified Deployment**	Deployments are truly environment agnostic.

Use Cases

1	**Sporadically used business logic, like an application that sees high usage only a few times a month**	You do not want to keep extra hardware on standby when it is only needed a few times per month, such as in a ticket selling application.
2	**Lightweight applications that perform straightforward tasks**	Sometimes, applications are so lightweight even a single server is overkill.
3	**Small shop IT that wants to avoid infrastructure management**	Generally, you want to focus on your product, not running a data center.

7. Serverless drawbacks:



Defining Serverless

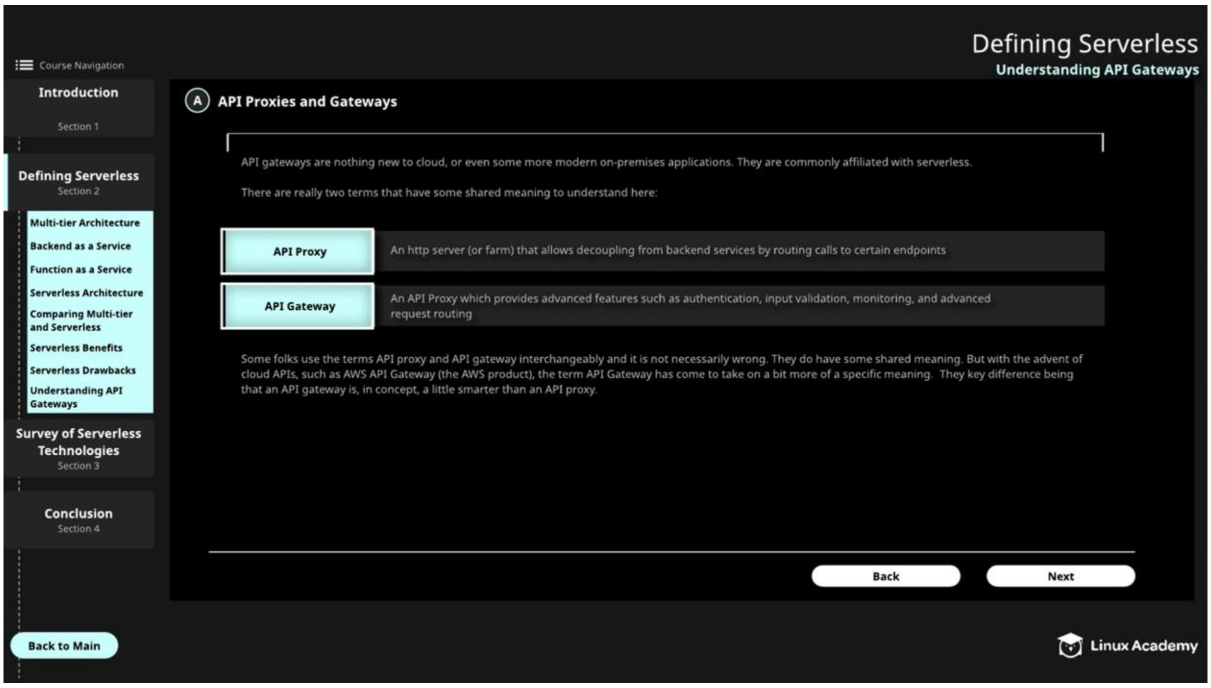
Serverless Drawbacks

Some Serverless Cautions:

```
Must keep less used applications warmed up:
    FaaS applications will be fully stopped if left idle too long!
Easy to become vendor locked:
    Each cloud provider does things a little different.
Multi-tenancy concerns:
    Data security can be a concern.
    Resource segmentation could also be a problem.
No ability for infrastructure optimization:
    You do not have a lot of intel on where your code is running.
Monitoring can be complicated:
    Each execution of code costs money even, if it is just a monitoring check.
    Logs are going to be your friend.
```

Your results may vary:
These disadvantages might be mitigated or even moot in certain use cases. You will need to evaluate the pros and cons for your situation.

8. Understanding API Gateways:



Defining Serverless

Understanding API Gateways

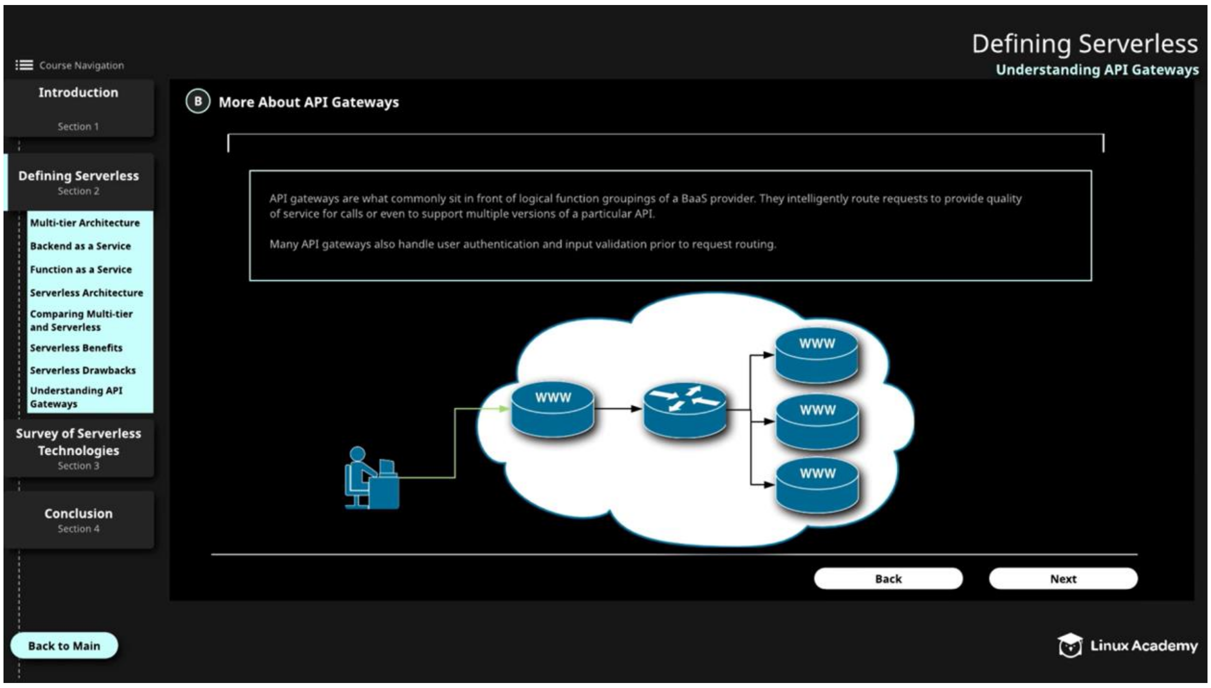
A API Proxies and Gateways

API gateways are nothing new to cloud, or even some more modern on-premises applications. They are commonly affiliated with serverless.

There are really two terms that have some shared meaning to understand here:

API Proxy	An http server (or farm) that allows decoupling from backend services by routing calls to certain endpoints
API Gateway	An API Proxy which provides advanced features such as authentication, input validation, monitoring, and advanced request routing

Some folks use the terms API proxy and API gateway interchangeably and it is not necessarily wrong. They do have some shared meaning. But with the advent of cloud APIs, such as AWS API Gateway (the AWS product), the term API Gateway has come to take on a bit more of a specific meaning. They key difference being that an API gateway is, in concept, a little smarter than an API proxy.



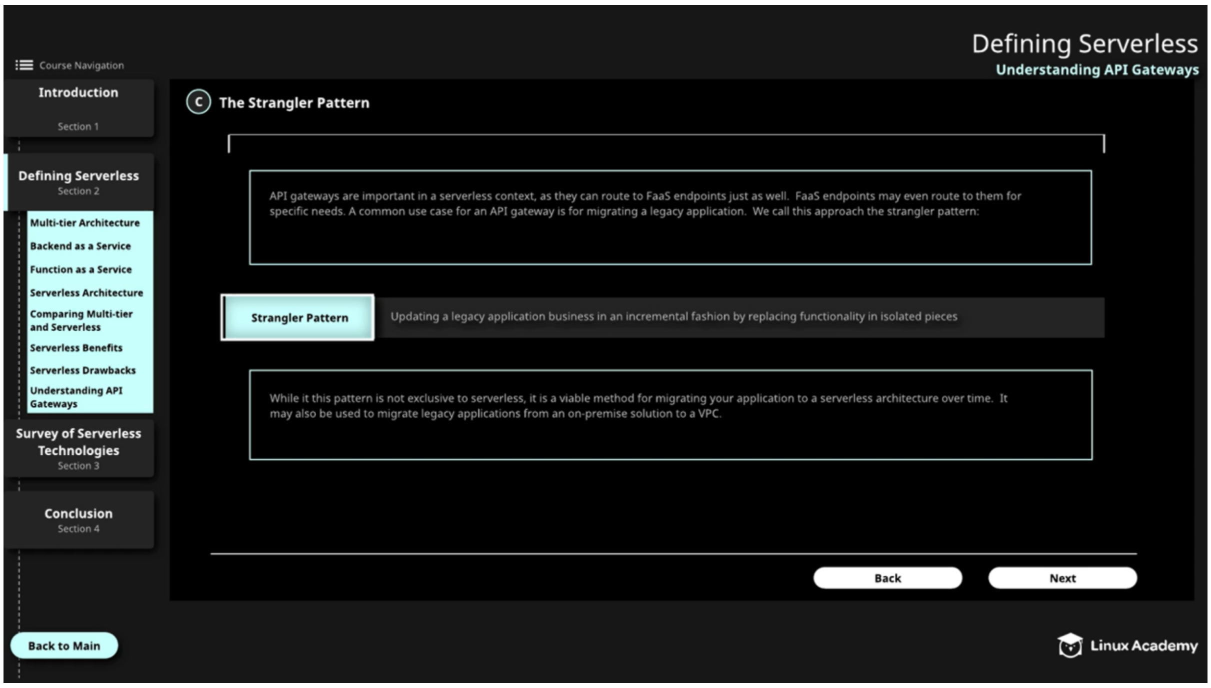
Defining Serverless

Understanding API Gateways

B More About API Gateways

API gateways are what commonly sit in front of logical function groupings of a BaaS provider. They intelligently route requests to provide quality of service for calls or even to support multiple versions of a particular API.

Many API gateways also handle user authentication and input validation prior to request routing.



Defining Serverless

Understanding API Gateways

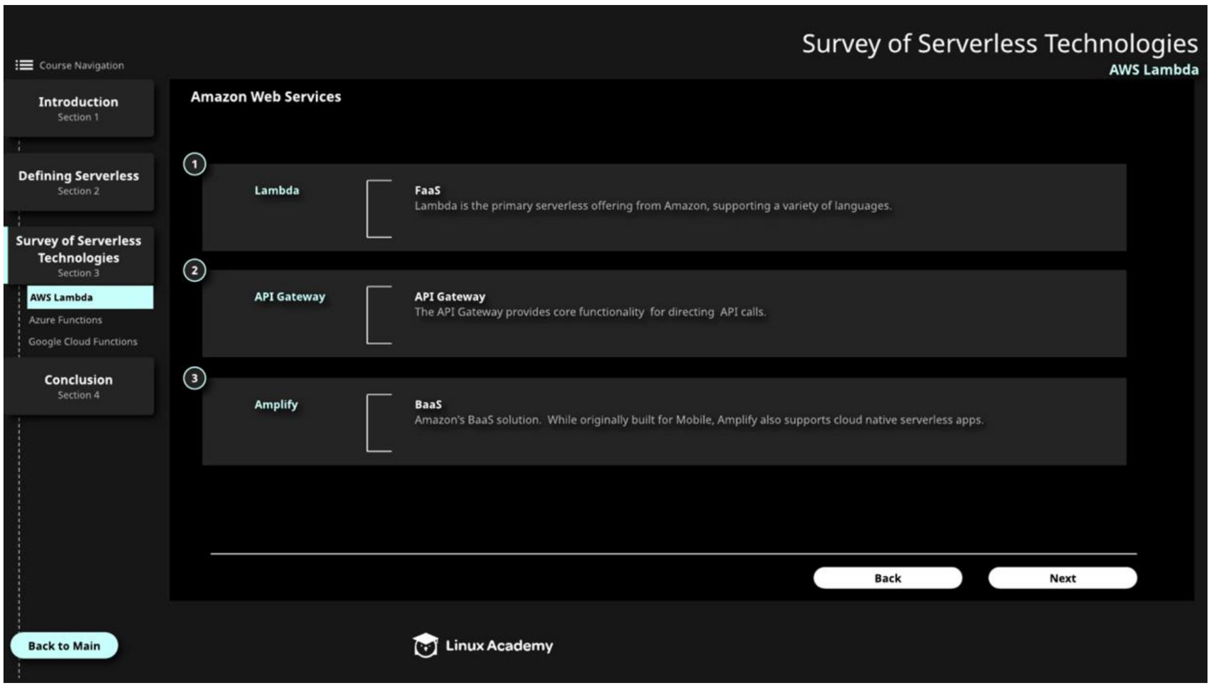
C The Strangler Pattern

API gateways are important in a serverless context, as they can route to FaaS endpoints just as well. FaaS endpoints may even route to them for specific needs. A common use case for an API gateway is for migrating a legacy application. We call this approach the strangler pattern:

Strangler Pattern	Updating a legacy application business in an incremental fashion by replacing functionality in isolated pieces

While it this pattern is not exclusive to serverless, it is a viable method for migrating your application to a serverless architecture over time. It may also be used to migrate legacy applications from an on-premise solution to a VPC.

9. AWS Lambda:



Survey of Serverless Technologies
AWS Lambda

Amazon Web Services

1. Lambda — **FaaS**
   Lambda is the primary serverless offering from Amazon, supporting a variety of languages.
2. API Gateway — **API Gateway**
   The API Gateway provides core functionality for directing API calls.
3. Amplify — **BaaS**
   Amazon's BaaS solution. While originally built for Mobile, Amplify also supports cloud native serverless apps.