Complete Guide to API Innovation, Design, and Management
In this e-guide:

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

APIs allow organizations new and old, large and small to drastically improve the speed and efficiency of their operations, but getting to this point with APIs can be difficult.

Read this E-Guide to learn our experts’ best practices for designing and managing APIs.
How the use of APIs helps spur business innovation

Zachary Flower, Freelance writer

APIs enable old and new organizations to change the way they do business. The use of APIs is powering the way forward, helping organizations expand old business applications and create new ones.

When architected properly, APIs give organizations the ability to drastically improve the speed and efficiency of their operations with minimal overhead and investment, which, in turn, opens doors to entirely new ways to run, grow and monetize a business. But what exactly does this process look like? And what does it mean to take an API-first approach to digital transformation initiatives?

Why API-first?

Innovation is an important step in the growth of any business. But what exactly is API-first innovation, and why is it the right path forward?

To take an API-first approach to digital transformation initiatives means to develop an API before tackling innovation. Countless industries use this approach through the use of accelerator programs, incubators, hackathons and developer competitions.

The hackathon tactic in particular is used by everyone from microbreweries to telecom companies and is an excellent way to generate new business cases using capabilities the organization already has at its disposal. Through the creation of APIs that expose functionality
and data that is core to their business, these companies can provide a platform for enterprising developers and innovators to build new, inventive applications and services on their behalf, while putting a limit on the risk involved with the development, validation and launch of new features.

But this is not just about growth. Because of their explicitly programmable nature, the use of APIs lends itself to the improvement and replacement of existing business processes. While some may interpret this to mean that parts of the business can be automated more easily, a more accurate interpretation is that organizations can improve the core architecture of a business by making the data that powers it more accessible. APIs empower organizations to gather and analyze metrics that include inventory counts and consumer demand. Through the creation of data-driven APIs, organizations can use this information to build a vast library of analytics. Businesses can use these analytics to understand previously undetected trends.

How to go API-first

Today, it is common for a company to consider itself a "digital" company, regardless of industry. But just because companies are digital-friendly doesn't mean they use APIs properly.

The most obvious way to take an API-first approach to innovation is to create an API on top of technologies or processes that already exist. This is an ideal approach since it puts the organization in total control over its data. Unfortunately, creating an API isn't necessarily in the wheelhouse of nontechnology companies. But through the use of third-party API management platforms, companies can create an API with almost no technical expertise.

While API creation is a great way to open new doors for any organization, I recommend you start with a focus on data. In a nonsoftware company, creating an API that exposes more than just data can be a difficult task that potentially requires the introduction of a new engineering
In this e-guide

- **How the use of APIs helps spur business innovation**
- **Best tools and methods for designing RESTful APIs**
- **Open source tools to consider for your RESTful APIs**
- **API integration becomes an enterprise priority**
- **How to adapt API management for serverless architecture**
- **How to use microservices to manage serverless APIs**

The use of APIs is a great, cost-effective way to innovate within a business. Real change is often not cheap. But the simplicity of data-driven APIs allows organizations to innovate with minimal overhead. Ultimately, an API-first approach to innovation provides a platform to identify potential business improvements and ensure that the organization evolves in an efficient and sustainable way.

There’s another path toward API-first innovation: consuming external APIs. While creating an API to foster innovation and improve transparency is a great way to grow a business with minimal investment, it isn't the only way. APIs are meant to be consumed; their creators intend for other organizations to use and benefit from their API.

The availability of these external APIs can be a powerful tool when organizations explore new ways to foster growth. Take, for example, a photography company that specializes in editing tourists out of vacation photos. While this is a mostly manual process, the introduction of an object-recognition API trained to identify people may speed up the process significantly. This relatively small example is a low-cost way to experiment with new technology without building it yourself.

The use of APIs is a great, cost-effective way to innovate within a business. Real change is often not cheap. But the simplicity of data-driven APIs allows organizations to innovate with minimal overhead. Ultimately, an API-first approach to innovation provides a platform to identify potential business improvements and ensure that the organization evolves in an efficient and sustainable way.

▼ Next Article
Best tools and methods for designing RESTful APIs

Tom Nolle, President

There are two broad pathways to designing RESTful APIs.

One is the open process approach, which is the most common. This approach focuses on steps and tools that support the process of API design, placing few, if any, constraints on the nature of the APIs to generate.

The second is the model approach -- also known as the specification approach -- which starts with a specification and/or model of a RESTful API and supports the adaptation of that model to the specific interface requirements of each project. As APIs become more common, this strategy is gaining ground, but not without its share of complications.

The open process approach

It's fairly easy to turn software architect-created Unified Modeling Language into an API specification, and most of the interactive development environments, such as NetBeans and Eclipse, have tools that facilitate a so-called open process transformation. These tools presume that the basic nature of the API is defined by the component relationships, and the translation of those relationships into the actual APIs is fairly mechanical, especially it's if a single API or a set of closely related APIs within a limited application scope.

The model approach
The goal of the model approach is to create best-practices structures for all RESTful APIs -- and sometimes even for all APIs -- in which the APIs are refined to lend themselves to a specific function. This approach enforces standard ways to do the same thing in different places, which reduces developer confusion and errors, as well as facilitates security and compliance monitoring.

### Five elements of API design tools

Both the open process and the model approaches use a multitude of tools to solve problems that result from designing RESTful APIs. These tools tend to address issues of API visualization, API cataloging, API model enforcement, API code generation and API testing. However, it's sometimes difficult to figure out which API design tools address which of these issues.

API visualization is one of the fundamental steps in design, because it frames a graphical view of the API for users and enables users to interact with services that use a type of generalized API GUI. Most interactive development environments have visualization tools available, but these tools only offer basic capabilities. Swagger UI is a popular API visualization example that makes the in/out data structure of an API visible; it also exhibits simulated responses to given API caller requests.

An API catalog is the central element of any API design strategy. Catalogs hold API definitions and make them available to developers. In some cases, catalogs may also drive API management processes, like access control or load balancing. Most API management suites will include a catalog, and separate API catalog tools are available from companies like Swagger, Oracle and IBM, as well as in open source form, like ReDoc. Catalog tools may provide a computer-browsable catalog, a printed API document set or both. Decide what to look for before choosing a specific tool to manage the catalog.
API model enforcement and standardization is at the heart of the model approach to designing RESTful APIs. These types of tools require a structured set of API definitions, using formats like Swagger, API Blueprint or RESTful API Modeling Language (RAML). In this critical element of the model approach to API design, it’s best to start with the API definition standards and then jump off to tools that support them. Otherwise, you may end up with a set of tools that don’t connect to each other, which would leave gaps that can hinder and defeat the implemented standards on the APIs.

API code generation is helpful to quickly convert a design to implementation without risk of errors. Many of these products are integrated with, or part of, a catalog solution. APImatic and REST United are examples of code generators, and Swagger also provides a similar API tool in its suite. The big advantage of code generators is their facilitation of rolling changes to software when an API changes.

Testing APIs should always be a part of an API design task and can include API virtualization as a technique. Most API management suites include some level of API testing, and both SmartBear and Runscope offer a range of API testing tools, which include sliding scale pricing depending on scope and features. SmartBear also provides monitoring capability. API monitoring helps keep track of API performance and health with analytics-based tools, like APImetrics.

**Tool selection approaches**

Certain API management suites have many useful tools for designing RESTful APIs. It’s worth looking at MuleSoft’s API management tools, as well as IBM’s Bluemix, Rogue Wave’s Akana and Swagger’s suite of API tools. Swagger probably offers the most complete set of API design and validation tools available, so there’s nothing wrong with starting a tool review there. Also, review the major cloud providers’ API tools; Amazon, Google and Microsoft all
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

offer tools that facilitate the use of their cloud services models, and some can extend to broader enterprise missions.

Secondly, there are open specifications and modeling languages for API design. Swagger’s OpenAPI Specification is widely used by API tool vendors, and it’s one of the best places to start a search for practical API design tools that work together. RAML and its associated tools present another good way to unify a selection of tools under a common model. If you don’t find a management suite that works, adopting the model approach is the next best option for unifying your API design. Visual Paradigm has a nice RESTful design tool that works with several of the specification and modeling products.

Sources like Moesif offer a complete set of API design patterns and best practices, which may be the best approach if you’re planning to adopt an open process approach to API design. Try to find design patterns that define a broad RESTful model, and then refine it for various types of APIs, rather than a disconnected set of process-specific design patterns.

In the long run, leaning toward a model-driven or specification-driven approach in API design is probably the best route, especially since APIs aren’t going to get simpler in a microservices-driven future. Even if you’re committed to the open process approach, try to use models and specifications to evolve your business’s API thinking and prepare for the future.
Open source tools to consider for your RESTful APIs

Twain Taylor, Freelancer

At the start of a RESTful API development project, a software team might be tempted to buy an expensive commercial API management tool when an open source tool can just as easily do the trick. In fact, there are plenty of open source tools that can help with each stage of the API lifecycle and help get an API development program off the ground at low cost.

Tools for RESTful API creation and management

Kong and Tyk are two popular providers that offer free community options for general API lifecycle management. Swagger also features several open source tools, such as Swagger Codegen, Swagger Editor and Swagger UI, which make it easy to build and define RESTful APIs.

RESTful API Modeling Language (RAML) is another API development platform that that lets software teams design RESTful API architecture visually without the need to write code, but it then uses server-side code generators to convert this design into code. Once built, RAML makes it easy to initiate test-driven development and perform unit tests on your RESTful APIs. It integrates with many API testing tools, lets you create documentation on the fly and makes it easy to integrate APIs with external systems.

Tools for RESTful API testing
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

After creation, it’s critical to conduct security and performance tests on your RESTful APIs. For this task, use a tool that focuses exclusively on API testing and documentation.

Insomnia is a great debugging tool for RESTful APIs. It helps developers test URLs, payload, authorization and headers. It eases debugging through a feature called environment variables that lets you specify repeating values and reduce manual entry.

Insomnia has a polished and efficient user interface plus autocomplete suggestions for selecting variables and operators for debugging. It supports several authentication types, like OAuth, AWS identity and access management and more. The free version includes a full-featured desktop app and is a great option if you don't need enterprise-grade security or multi-device access. Meanwhile, the paid version offers end-to-end encryption and unlimited device access.

Tools for RESTful API documentation

If you want RESTful APIs to gain widespread adoption and acceptance by third-party developers, make sure there is mature documentation. Great documentation helps developers understand the architecture of the REST API and makes it easy to build services that communicate with the API quickly.

Socket is a great option for documentation. Socket's primary commercial offering is an API marketplace, which features a wide range of APIs ready for integration. As a side gig, Socket created a free API documentation tool to help developers document their APIs, test them and then ultimately integrate them within applications.

And if it aligns with your API goals, Socket is a great platform for promotion and adoption. The only catch is that Socket makes you list your API on their marketplace. However, this could be a bonus if you plan to build a public API and want to drive usage and adoption.
RESTful APIs are the language of the modern web, and an effective API strategy focuses on API design, development, testing and documentation. Fortunately, there is no need to buy expensive commercial tools to design and develop RESTful APIs. These open source tools can help you build RESTful APIs that are a breeze to integrate with and enable powerful user experiences.
API integration becomes an enterprise priority

Jan Stafford, Features Writer

More organizations are looking for strategies that enable faster -- and cheaper -- connections between internal and external multi-cloud, hybrid cloud and on-premises application environments. Experts say API integration and event-driven integration will play a key part in those evolving strategies.

Widespread adoption of cloud and mobile applications creates an urgent need to connect standalone applications and services. APIs come in handy here, as each API provides code that enables communication between software programs. API integration is one way to connect internal enterprise cloud, mobile and on-premises applications, as well as with partners' and customers' application environments.

Investments in integration technology platforms, which include software licensing, maintenance and cloud services subscriptions, reached $12 billion in 2017. This marks a double-digit growth spurt from the previous year, said Massimo Pezzini, a research fellow at Gartner. By 2022, revenue is predicted to hit $19 billion.

The fastest-growing segment is API integration technologies, which are typically delivered via API management, integration platform as a service (iPaaS) and hybrid integration platforms. While SMBs will use the first two of those platforms most often, at least 65% of the largest organizations will implement a hybrid integration platform by 2022.

Spending on traditional point-to-point enterprise application integration platforms will see only single-digit percentage growth for the next five years, Pezzini said. Even so, don't expect a
mass exodus from legacy integration platforms, said Forrester analyst Randy Heffner. Forrester's research shows that only 16% of developers who work in cloud computing environments regularly use public cloud-based integration products. Of that 16%, only 45% say they run or intend to run more than 90% of their integration flows in a cloud-based integration platform.

Most of these developers will augment traditional integration practices with pre-built integrations delivered in API management platforms.

"One should have strong architectural layering between the two," Heffner said. "Plenty of firms have both -- and the layering is a best practice -- even if there is some incidental overlap between selected products."

### Why API integration?

"As new applications come in and are being introduced to consumers, adoption can skyrocket quickly to tens of millions users," said Matthew David, emerging technologies digital product owner at QBE Insurance. "With API integration, your delivery team can react quickly to make your applications available to them."

While API integration is new to some companies, the technology behind it is seasoned and mature, David said. The technology in APIs has its history in web services, so there's solid API technology that allows you to create services that you can share across systems. Now, API management platforms can automate API integration tasks, create API catalogs and more.
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

Matthew David

The pre-built integration templates in API integration products bring quick connectivity between previously siloed cloud applications. These packaged integrations also help with self-service deployment for line-of-business employees, increasing the speed and reducing labor costs of integration.

Those attributes led Humantelligence, which offers an AI-driven recruiting and culture-analytics platform, to adopt API integration. Juan Luis Betancourt, Humantelligence’s CEO, sought automated integration capabilities to connect the company’s app environments with customers’ cloud and homegrown apps, particularly their applicant-tracking applications.

After evaluating five products, Betancourt implemented Jitterbit Harmony iPaaS. This API integration platform helps his company quickly connect SaaS, on-premises and cloud applications. "The iPaaS solution provides the built-in integrations and automated tools we need to navigate the complexities of API integration," he said.

Without API integration, Betancourt’s team had to build integrations with customers’ homegrown on-premises and new cloud applications. With a staff of only five developers, doing just one such integration projection would freeze company development and certain
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

operations for two weeks or more. Customer-support improvements, feature-release planning and other critical functions had to be put on hold.

"It was really crippling for us to do integrations ourselves," Betancourt said.

**Extend APIs to event-driven integration**

About 50% of managed APIs will support event-driven IT by 2020, according to Gartner. These event-driven APIs are part of the technology mix that will help a business achieve a competitive advantage with real-time integration. "Event-driven integration technologies help businesses sense and respond quickly to what’s going on with customers, the industry [and] the market," Pezzini said. Event-driven APIs will supplement and not replace RESTful API request-and-response technologies.

Developers will continue to build increasingly event-driven API models, notably using a callback model that gives an API an address for where to deliver a result.

"Being able to handle business events more effectively and more expediently is a critical process in solution architectures," Heffner said.
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

In event-driven integration, systems communicate to recognize events and respond by instigating actions in other systems and applications.

A typical example is closing an airplane door, which Chatterjee called "both a business and a technical event." Via integration with systems' APIs within the airline, air-traffic control and other entities concerned, this event can trigger delivery of data about flight timelines.

"When a business event gets captured, it immediately triggers the business rules through an API," Chatterjee said.

If there's a flight delay, for example, the event signals APIs for the re-scheduling and re-rostering of staff on connecting flights -- activities that will get pushed to central computing and then to notification devices, such as smartphones and tablets. This was done with messaging services in the past, but can be more effectively done via API-based systems, complex event processing engines, in-memory data and messaging services.

Integration is a transformation project
As businesses craft more flexible application integration strategies, they should recognize that integration is not just a technology play. Instead, an enterprise integration strategy is part of businesses’ digital transformation initiatives.

"Look at how to use integration capabilities within the organization to fuel growth and transformation rather than being more like an underpinning of technology," Chatterjee said.
How to adapt API management for serverless architecture

Chris Moyer, VP of Technology

If you’ve already built a stable API infrastructure, you don’t want to just throw everything away and start over. After all, the term legacy is often another way of referring to things that already work for you. But how can you adapt an existing API to a serverless architecture?

Many tools exist to help convert APIs piece by piece, enabling you to monitor and replace individual parts of the APIs as you transition to a serverless architecture.

Knowing where to start

A traditional API system is built on layers. Therefore, you might have an authentication piece that checks credentials against something like Active Directory (AD), a query interface that converts user input into SQL. You might also have middleware that, for example, processes raw HTTP headers and converts them into formats specific to your system. You can transition each piece to a serverless architecture, but you’ll want to create a set of tests that make sure they transition properly.

API transitions are perfect to use for test-driven development. To begin, you should try to replace the least complicated APIs. These should be the simplest to test, and this will give you a handle on API migration. It’s also important to make sure you can easily roll back a change and revert to the old API version if needed. For example, you don't want to update the domain name system (DNS) records with a time-to-live (TTL) of one week, because that means it could take up to a week to update and roll back a change. Instead, consider lowering
TTLs for the DNS and setting transitions to something like five minutes. You could also use an API proxy or load balancer to route traffic as needed.

One common migration approach is to replicate traffic to a new API while disabling any side effects -- such as charging a credit card -- so you can test the new API with real user data. Sending identical data to both the old and new API enables you to verify that the output of both APIs are compatible, and it lets you identify issues that the switch might cause.

The next step is traffic shifting or A/B routing, which can be carried out through an API gateway. Have the gateway send most of the traffic to the old API and a smaller percentage of traffic to the new API. Monitor for unusual activity, and watch for user complaints, which will help verify that the new API works without customer impact.

**Versioning APIs**

If you have public APIs, it's always a good idea to version them. If you work with an old SOAP API but want to offer up an event-based REST API, you can offer both simultaneously for a period of time while users transition to the new API. API versioning typically involves the addition of a version prefix to a URL, such as `mycompanyapi.com/v1/users`. But developers can also create separate endpoints, such as `v1.mycompanyapi.com/users`. Different formats of API responses typically use suffixes, such as `users.json` or `users.xml`. The most important thing is to communicate the transition to end users and let them test out the new APIs, but still provide access to the old APIs in case something goes wrong.

It's also important to install monitoring that identifies any users or internal applications that still use the old APIs. It's generally good practice to provide notice at least six months to a year before an old API is terminated, provided there aren't any critical security issues.
Creating adapters

Exposed APIs don't necessarily need to change, even when the underlying architecture changes. You can still move an API to a serverless architecture and not modify the user-exposed API when the implementation changes. Even if the API exposes a WebSocket, you can run a separate adapter on an Amazon Elastic Compute Cloud (EC2) instance. You can also run it through Amazon EC2 Container Service using the Fargate launch type, which simply exposes the WebSocket API but uses the new serverless API under the hood.

Adapters on top of an API provide users with a simple way to access the API in a familiar way. For example, if your API only produces JSON outputs but your users want a SOAP response, you could create a specialized SOAP adapter that converts requests from the SOAP format to the new JSON REST format.

What should I scrap?

There will always be certain pieces of an old API architecture that won't properly convert to serverless, such as lock files or local files. Anything that requires sequential ordering, such as an automated ID that increments by one with every new purchase, does not work well with web-scale databases. On the other hand, elements like single sign-on (SSO) are a good fit for serverless.

If your authentication process jams up the works because it only allows six-character passwords, it's time to scrap that. Authentication and...
authorization are the most important parts of any API, and they can also be the biggest hindrance. Caching can help, but if the actual login verification process takes a few minutes, your API is essentially broken. Consider swapping out an old authentication service like AD for a modern SSO system. Or integrate a social media login that enables customers to easily sign up and log in.

Not everything needs to be migrated to serverless

The most important thing to remember during a serverless migration is that not everything needs to migrate. If 90% of your requests are designed to constantly read a list of products, it makes sense to migrate to serverless. If you only create one new product a month, it does not make sense to migrate to serverless. Instead, create a back-end SQL database that synchronizes changes to Amazon DynamoDB in a format that the serverless API can read, but not write to.
How to use microservices to manage serverless APIs

Chris Moyer, VP of Technology

As organizations of all types continue to kick their own API development into high gear, some developers may find their old monolithic services are simply too heavy to provision using something like AWS CloudFormation. This is a sign it’s time to break your service into multiple smaller services that don't contain large numbers of API endpoints. Let’s review how and why microservices can help you create and manage serverless APIs.

The Serverless Framework error

Serverless Framework is a popular API creation and management tool on AWS Lambda. It connects to API Gateway and Lambda through a CloudFront template, which makes it easy to write an application once and then release it to staging and production environments. But Serverless Framework can display an error that can scare developers into thinking everything they’re doing is wrong:

The CloudFormation template is invalid: Template format error: Number of resources, 201, is greater than maximum allowed, 200

This error message can confuse developers who don't have over 200 API endpoints. For example, a user can have about 50 API endpoints and still encounter this message.

This misunderstanding occurs because resource does not mean API endpoint. An API endpoint might look like a single resource while it is in Serverless Framework, but once
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

compiled to the CloudFormation template, that endpoint may be split into three or four different resources: one resource for the API Gateway connection, one for the Lambda function, one for an associated identity and access management role and possibly another if there are Simple Queue Service or Simple Notification Service topics also associated with the API endpoint.

There's no way to break down the endpoints of these serverless APIs or split them into multiple stages of deployment, so developers are essentially stuck re-architecting their API.

**Splitting the serverless monolith into microservices**

Knowing that a single CloudFormation stack can't really contain more than a few API endpoints, developers must plan to split that service into multiple services. Many developers do this already, so why not just split into microservices and keep everything separate? Developers can even split out common tasks, such as user authentication, into entirely separate Lambda functions that can be called by any of the separately created microservices.

For example, a company may have one serverless service that fetches a license by an ID, a second that updates a license by an ID and a third that fetches account information. All of these services share the common task of authenticating and validating user permissions for API access.

Since each of these serverless APIs is a completely separate serverless service, they each create their own CloudFormation stack and API Gateway. As a result, the developers can't simply create an authorizer function in one of them and share them across each of their microservices. Instead, they split that authorizer function out into its own microservice.
In this example, the authorizer function actually sits in its own separate repository, and it isn't even included with its own serverless service. It's simply a Lambda function that builds using CodeBuild with a very basic buildspec.yml configuration:

```
version: 0.2
phases:
  install:
    commands:
      # Pre-install dependencies and environment config
      - apt-get update
      - apt-get install -y ssh
      # Download SSH keys from S3
      - aws s3 sync s3://company-config/home/ ~/
      - chmod 0600 ~/.ssh/*
      - npm install npm@6 --global
      # Install dependencies needed for running tests
      - npm install
  pre_build:
    commands:
```

Page 24 of 29
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

# Discover and run unit tests in the 'tests' directory

- npm test

build:

commands:

- # Build the js files
  - ./node_modules/.bin/tsc

- # Zip everything up excluding the node_modules/@types directory
  - zip -r archive.zip
    *.js src/*.js node_modules
    -x "node_modules/@types/*"

- # Use the AWS CLI directly to upload our lambda function
  - aws lambda update-function-code
    --function-name authenticate
    --zip-file fileb://archive.zip
    --publish > ./resp.json
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

# Tag the new version

```bash
aws lambda update-alias
--function-name authenticate
--name $BUILD_ENV
--function-version
$(echo "console.log(require('./resp.json').Version")|node)
```

This basic auto-build script lets you avoid the unnecessary overhead of using a framework, like Serverless or Apex, and lets you link directly to the Lambda function from your other microservices. By using AWS Command Line directly, you're able to upload a new version of your microservice and tag it based on the build environment, or even tag specific releases by version if you choose. Because the AWS serverless APIs allow you to invoke a Lambda function by a specific alias, you can also maintain a separate version of your code for development, staging and production by setting the Qualifier property on lambda.invoke:

```javascript
const lambda_resp = await lambda.invoke(
    {
        FunctionName: 'authenticate',
        Qualifier: process.env.BUILD_ENV,
        Payload: JSON.stringify(event),
    }).promise();
```
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

Take note that this method does not automatically manage the IAM role and environment variables. The IAM roles should not change frequently for most microservices, so this hopefully shouldn't cause too much of an issue.

### Putting all of your microservices under one domain

Now that you have your separate serverless services to fetch licenses, update licenses and fetch user information, the next challenge is to expose all of those services to a single API domain. Using the Serverless Framework plug-in `serverless-domain-manager`, you can configure a custom domain that references each of your microservices under a different base path. Here's a licenses service example:

```json
custom:
  customDomain:
    domainName: api.example.com
    basePath: licenses
```

Unfortunately, the `basePath` cannot contain slashes, so developers must use separate domain names for each build environment by adding in the stage to the `domainName`:

```json
domainName: api-${opt:stage, self:provider.stage}.example.com
```

In this example, each stage will fall under its own separate domain, such as `api-dev.example.com/licenses`. To update a license, however, the API can't live under the same base path, which means the list, fetch and update operations all must exist under the same serverless service -- if you use the Serverless Framework. Alternatively, you can create each microservice as a simple Lambda function with qualifiers and create an API Gateway.
manually to link to them. By creating the API Gateway separately from the Lambda function, you can more easily deploy updates to individual microservices and then link them all under a single API Gateway and custom domain.

**Serverless Framework is the glue for your microservices**

The problem with Serverless Framework is that it doesn't simplify the process of splitting Lambda functions into separate microservices that deploy individually. Whenever you execute `sls deploy`, it bundles all the code in the current directory and uploads it as a CloudFormation stack. This means that every Lambda function in that service uses the same code base with a different entry point. That configuration violates the core idea of microservices, which is to have each microservice run isolated and independent of each other.

It's also troublesome to split each microservice into its own serverless service, as you can't have multiple microservices exist under a single API Gateway base path. To use serverless APIs with microservices, you should keep each API Gateway separate from each other and produce a separate CloudFormation Stack for every microservice. This is not ideal and makes you question using Serverless Framework with microservices at all. However, the better approach is to use Serverless Framework to deploy the "glue code" which connects each of the deployed microservices separately under one single API Gateway.
In this e-guide

- How the use of APIs helps spur business innovation
- Best tools and methods for designing RESTful APIs
- Open source tools to consider for your RESTful APIs
- API integration becomes an enterprise priority
- How to adapt API management for serverless architecture
- How to use microservices to manage serverless APIs

Open source tools to consider for your RESTful APIs
API integration becomes an enterprise priority
How to adapt API management for serverless architecture
How to use microservices to manage serverless APIs