Migrating a legacy web application to an API-based architecture on Azure

09/13/2018 • 4 minutes to read • Contributors all

In this article

An e-commerce company in the travel industry is modernizing their legacy browser-based software stack. While their existing stack is mostly monolithic, some SOAP-based HTTP services exist from a recent project. They are considering the creation of additional revenue streams to monetize some of the internal intellectual property that's been developed.

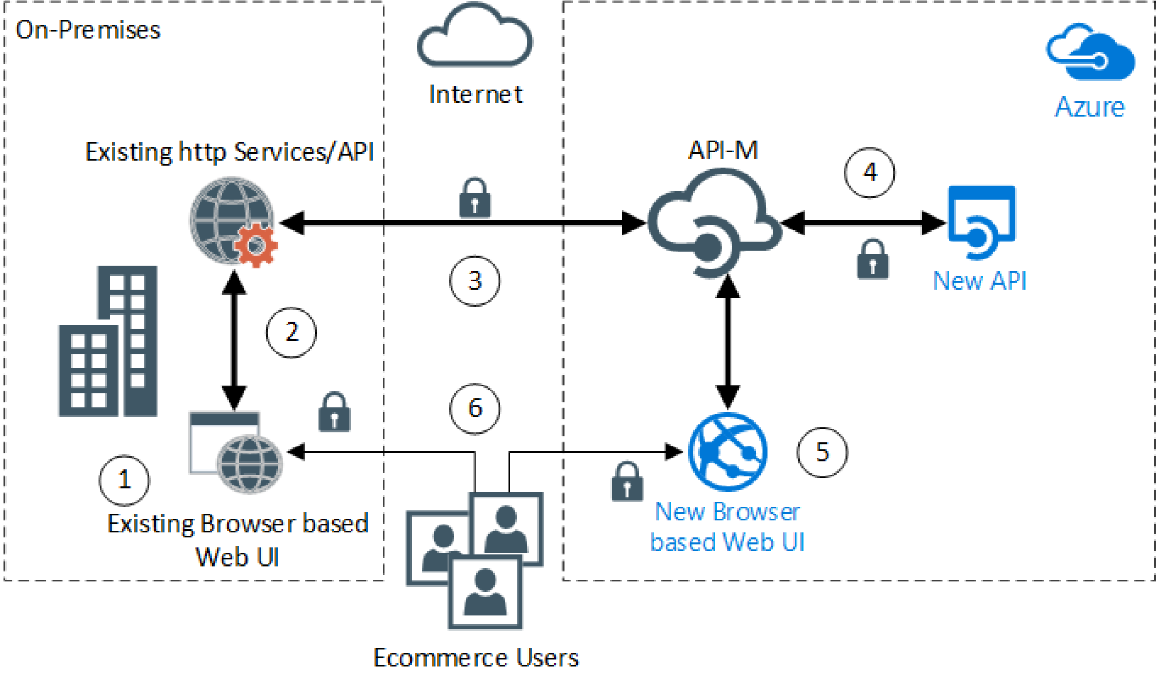
Goals for the project include addressing technical debt, improving ongoing maintenance, and accelerating feature development with fewer regression bugs. The project will use an iterative process to avoid risk, with some steps performed in parallel:

- The development team will modernize the application back end, which is composed of relational databases hosted on VMs.
- The in-house development team will write new business functionality that will be exposed over new HTTP APIs.
- A contract development team will build a new browser-based UI, which will be hosted in Azure.

New application features will be delivered in stages. These features will gradually replace the existing browser-based client-server UI functionality (hosted on-premises) that powers their e-commerce business today.

The management team does not want to modernize unnecessarily. They also want to maintain control of scope and costs. To do this, they have decided to preserve their existing SOAP HTTP services. They also intend to minimize changes to the existing UI. Azure API Management (APIM) can be used to address many of the project's requirements and constraints.

Architecture



The new UI will be hosted as a platform as a service (PaaS) application on Azure, and will depend on both existing and new HTTP APIs. These APIs will ship with a better-designed set of interfaces enabling better performance, easier integration, and future extensibility.

Components and Security

1. The existing on-premises web application will continue to directly consume the existing on-premises web services.
2. Calls from the existing web app to the existing HTTP services will remain unchanged. These calls are internal to the corporate network.
3. Inbound calls are made from Azure to the existing internal services:
    - The security team allows traffic from the APIM instance to pass through the corporate firewall to the existing on-premises services using secure transport (HTTPS/SSL).
    - The operations team will allow inbound calls to the services only from the APIM instance. This requirement is met by white-listing the IP address of the APIM instance within the corporate network perimeter.
    - A new module is configured into the on-premises HTTP services request pipeline (to act on **only** those connections originating externally), which will validate a certificate which APIM will provide.
4. The new API:
    - Is surfaced only through the APIM instance, which will provide the API facade. The new API won't be accessed directly.
    - Is developed and published as an Azure PaaS Web API App.
    - Is white-listed (via Web App settings) to accept only the APIM VIP.
    - Is hosted in Azure Web Apps with Secure Transport/SSL turned on.
    - Has authorization enabled, provided by the Azure App Service using Azure Active Directory and OAuth 2.
5. The new browser-based web application will depend on the Azure API Management instance for **both** the existing HTTP API and the new API.

The APIM instance will be configured to map the legacy HTTP services to a new API contract. By doing this, the new Web UI is unaware of the integration with a set of legacy services/APIs and new APIs. In the future, the project team will gradually port functionality to the new APIs and retire the original services. These changes will be handled within APIM configuration, leaving the front-end UI unaffected and avoiding redevelopment work.

Alternatives

- If the organization was planning to move their infrastructure entirely to Azure, including the VMs hosting the legacy applications, then APIM would still be a great option since it can act as a facade for any addressable HTTP endpoint.
- If the customer had decided to keep the existing endpoints private and not expose them publicly, their API Management instance could be linked to an Azure Virtual Network (VNet):
  - In an Azure "lift and shift" scenario linked to their deployed Azure virtual network, the customer could directly address the back-end service through private IP addresses.
  - In the on-premises scenario, the API Management instance could reach back to the internal service privately via an Azure VPN gateway and site-to-site IPSec VPN connection or ExpressRoute making this a hybrid Azure and on-premises scenario.
- The API Management instance can be kept private by deploying the API Management instance in Internal mode. The deployment could then be used with an Azure Application Gateway to enable public access for some APIs while others remain internal. For more information, see Connecting APIM in internal mode to a VNet.

> **Note**
>
> For general information on connecting API Management to a VNet, see here.

Availability and scalability

- Azure API Management can be scaled out by choosing a pricing tier and then adding units.
- Scaling also happen automatically with auto scaling.
- Deploying across multiple regions will enable fail over options and can be done in the Premium tier.
- Consider Integrating with Azure Application Insights, which also surfaces metrics through Azure Monitor for monitoring.

Deploy the scenario

To get started, create an Azure API Management instance in the portal.

Alternatively, you can choose from an existing Azure Resource Manager quickstart template that aligns to your specific use case.

Pricing

API Management is offered in four tiers: developer, basic, standard, and premium. You can find detailed guidance on the difference in these tiers at the Azure API Management pricing guidance here.

Customers can scale API Management by adding and removing units. Each unit has capacity that depends on its tier.

> **Note**
>
> The Developer tier can be used for evaluation of the API Management features. The Developer tier should not be used for production.

To view projected costs and customize to your deployment needs, you can modify the number of scale units and App Service instances in the Azure Pricing Calculator.

Related resources

Review the extensive Azure API Management documentation and reference articles.