opentext™ | Cybersecurity

Position Paper

Configuration Considerations for NetIQ Secure API Manager

NetIQ Access Manager customers now have the opportunity to extend their coverage to APIs. This position paper offers configuration considerations for those assessing how NetIQ Secure API Manager may benefit their environment.

Table of Contents

Configuration Considerations for NetIQ Secure API Manager

As microservices continues to expand its footprint across the enterprise, the need for a new approach to speed and security becomes more imperative. NetIQ Secure API Manager extends the power of NetIQ Access Manager to APIs.

The way enterprise applications are designed, developed, delivered, and maintained is rapidly evolving as the methods used to deliver extreme-scale, public-facing internet applications are adopted by enterprise architects. A number of terms are used to describe these methods, such as “Micro Services,” “Service Mesh,” and “Services-Based Architecture.” They all refer to an architecture that has the following basic properties:

- **Modularity:** Functionality is implemented as discrete components that have well-defined interfaces that allow for each type of component to be managed independently of other component types.
- **Composability:** Components can be easily combined in new ways as additional business functionality is required. Reusability and flexibility are inherent in the design of the components.
- **Scalability:** Components are designed and managed so that additional instances of a component can be deployed as needed.

“Micro Services,” “Service Mesh,” and “Services-Based Architecture” all refer to an architecture that has the following basic properties:

- Modularity
- Composability
- Scalability

Inter-Application Interaction Moves to the Network

All the possible details of this type of architecture is beyond the scope of this paper, so we will describe a very simple example application that illustrates some of the challenges involved and how NetIQ Secure API Manager by OpenText can be used to overcome these challenges. Let’s start by further defining what a component is and describing some of the common categories of components.

A “component” is a self-contained process or element of business logic that could be running on virtually any platform or runtime. It is very common for new components to be implemented within a container or “serverless” framework. But, especially in the Enterprise environment, components are often built by exposing functionality from existing systems or by reusing existing services.

An “application” in this context is the collection of all the components required to deliver the business functionality to the end user. From their perspective. it might seem like a single, monolithic entity, but in reality it is composed of a number of components. Each component could be part of multiple applications. For example, a component that enables you to look up the current price of a company’s stock could be reused in many applications. This type of application is sometimes called a composite application, since it is composed of independent, discrete components.

The key component for a composite application is the “front-end client,” or user interface. This can be a web application, a mobile app, or even a platform-specific client. Because the bulk of the application’s functionality and business logic is implemented by the back-end service components, this front-end client can be made remarkably simple. This simplicity makes maintenance of the front-end client easier.

The term “service” or “microservice” describes a component that is small, independent, and loosely coupled with other components. It performs a well-defined task and provides services through a documented and managed API. Most microservices are accessed over standard HTTP(S) network protocols. Because they are small and independent, they can be implemented and managed in ways that aren’t practical in traditional monolithic applications. Each microservice can be modified and scaled independently of other components of the application. The acronym “API” (Application Programming Interface) is often used as a generic reference to these services.

The term “service” or “microservice” describes a component that is small, independent, and loosely coupled with other components. It performs a well-defined task and provides services through a documented and managed API.

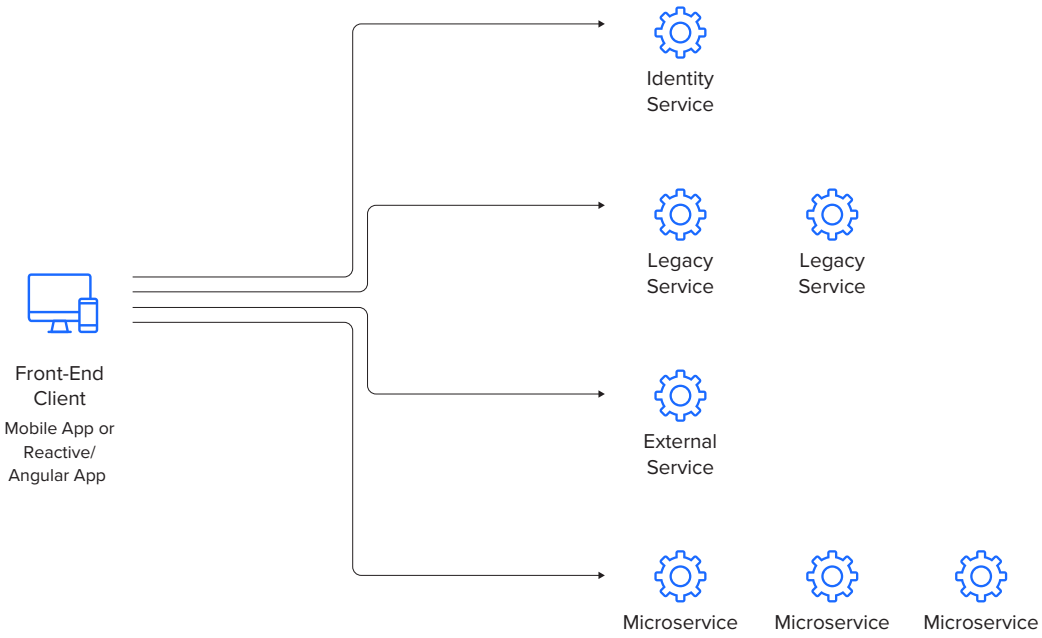

Figure 1. The use of purpose-built applications that interact directly with information sources create an access and security nightmare.

One way to implement a service-based application would be to simply create a front-end client (either a reactive web app or a mobile app) that makes calls directly to the services needed by the application. In the example on the previous page, the application utilizes legacy components, micro-services, and an externally provided service. It also uses an Identity Service to authenticate users and provide information about them.

One way to implement a service-based application would be to simply create a front-end client (either a reactive web app or a mobile app) that makes calls directly to the services needed by the application.

While this approach can certainly work, it has a number of issues:

1. Each service might utilize a totally different API technology: some could be SOAP, while others might be REST.
2. Each of the services likely has its own authentication and authorization implementation, so the front-end application would need to handle the complexity of authenticating with each one.
3. When the front-end client is the only point where the called services are all tied together, it becomes very difficult to effectively monitor the performance of each application and trace and diagnose issues.
4. Dealing with all this complexity in the front-end client means that developing, testing, and maintaining it becomes exponentially more difficult. Each time there is change anywhere, the client would need to be updated.

These issues, and many others related to this type of implementation, will prevent you from fully benefiting from a services-based architecture.

API Broker for Speed and Security

There are a number of approaches for resolving these issues and for reducing the complexity of the front-end client. The primary solution is to add an API management gateway, or broker component. The gateway will be used to:

- Consolidate calls to the back-end services.
- Take care of the differences in service implementation technologies.
- Enhance the ability to centrally monitor, troubleshoot, and secure the application.

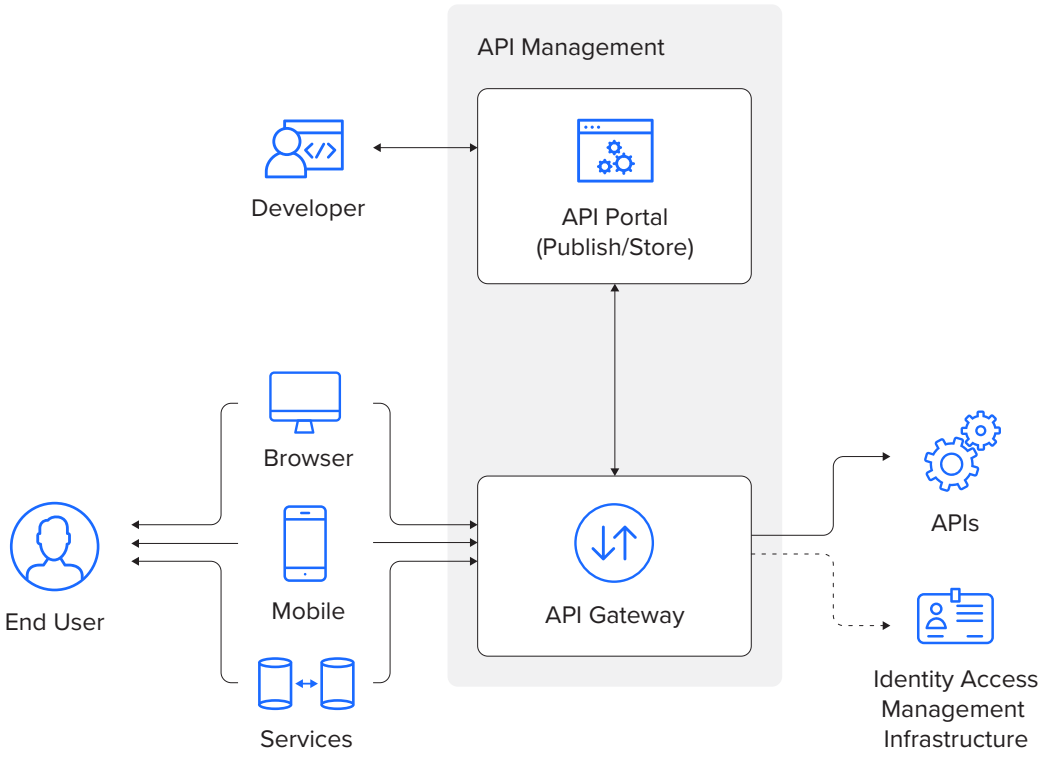

Figure 2. Boilerplate configuration of NetIQ Secure API Manager

NetIQ Secure API Manager (API Manager) by OpenText is an API management appliance that is integrated with NetIQ Access Manager by OpenText to provide API management capabilities that take full advantage of the robust authentication and authorization capabilities provided by NetIQ Access Manager. The same infrastructure that is used to protect your existing web applications can now be extended to protect APIs. Your existing federated authentication can also be utilized. And with NetIQ Advanced Authentication by OpenText, you have the capability to use strong, multi-factor authentication with your service-based applications.

NetIQ Secure API Manager (API Manager) is an API management appliance that is integrated with NetIQ Access Manager by OpenText to provide API management capabilities that take full advantage of the robust authentication and authorization capabilities provided by NetIQ Access Manager.

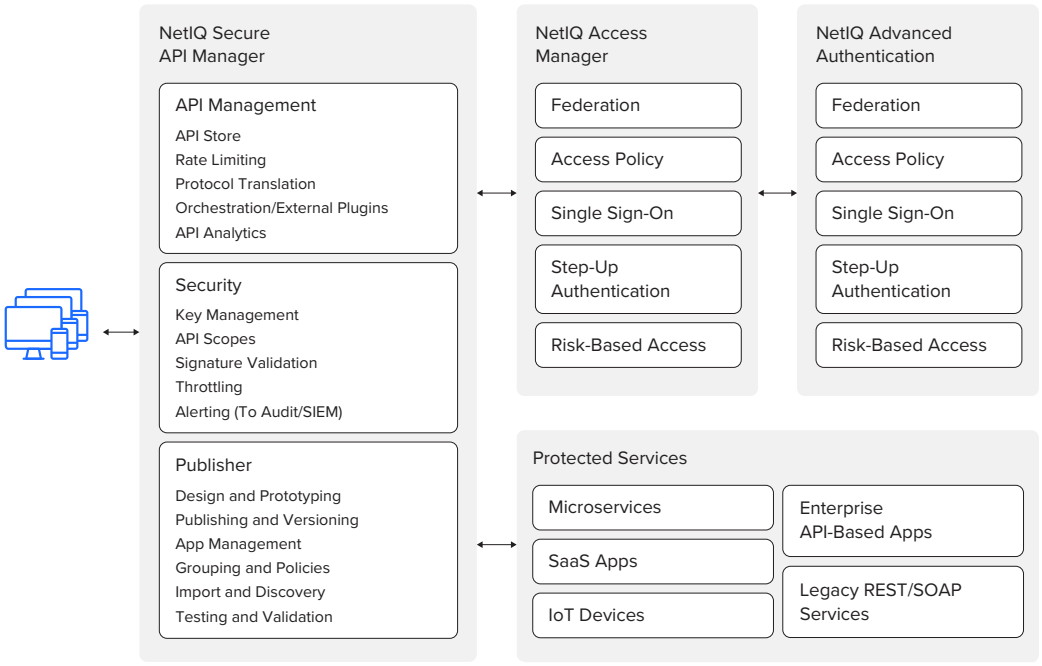

Figure 3. Components of NetIQ Access Management for APIs

The two major functional areas of NetIQ Secure API Manager are:

- API Gateway
- Lifecycle Manager

NetIQ Secure API Manager consists of two major functional areas:

- The **API Gateway** provides the runtime functionality to processes service requests. It enforces security, manages and limits API usage, and transforms requests and responses to and from the back-end services. As it does this, it collects data for monitoring and for analytics of API usage.
- The **Lifecycle Manager** is where APIs are implemented and managed. The Lifecycle Manager handles the publication of new services, controls updates to existing services, and, importantly, enables you to manage API retirement. Lifecycle Manager also collects extensive data for monitoring and analytics.

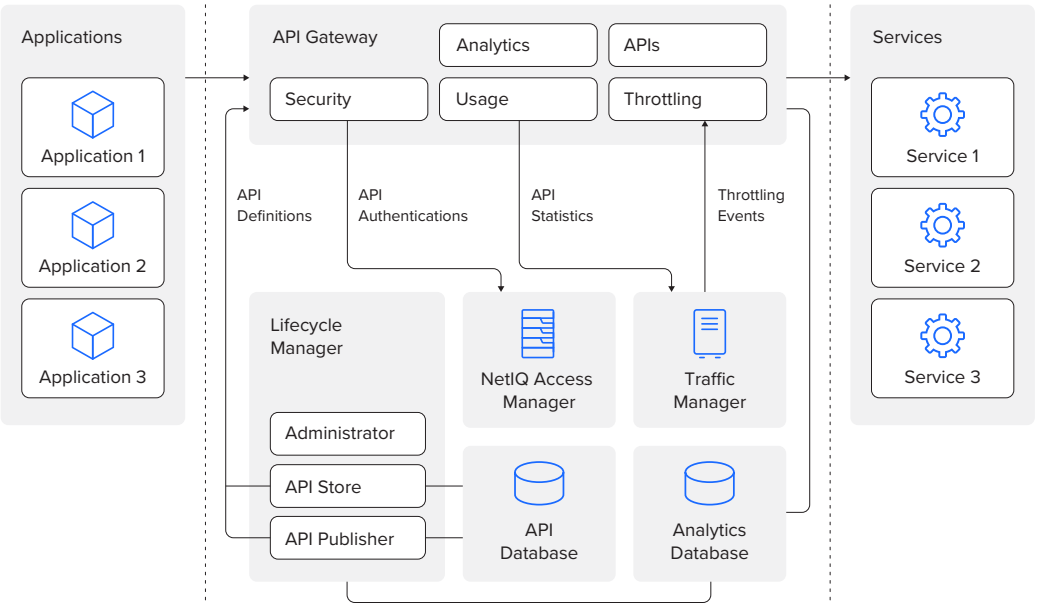

Figure 4. Functional breakout of NetIQ Secure API Manager

Notice that there is no separate application server dedicated to this application. The web application is served as completely static content that can be served from any web platform or even deployed locally. All of the display and business logic is built into the services and the front-end client.

Deployment Observations

The example we will utilize is a Call Center application that enables the call center worker to see a consolidated summary of information about a caller account. The information is retrieved from services provided by NetIQ Identity Manager by OpenText, ServiceNow, and a custom microservice that retrieves information from a legacy database. The application also enables the worker to send an SMS message to the user, using the Twilio SMS service.

Notice that there is no separate application server dedicated to this application. The web application is served as completely static content that can be served from any web platform or even deployed locally. All of the display and business logic is built into the services and the front-end client. Because of the “serverless” design, this application is easily scaled by adding additional gateway and service nodes. The design also provides extremely flexible options for fault tolerance and disaster recovery.

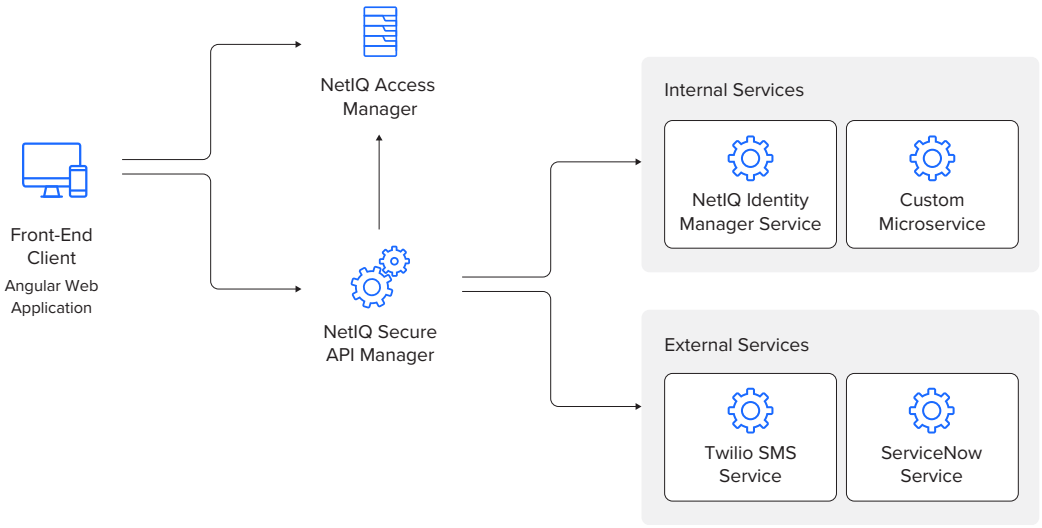

Figure 5. Call Center example

Secure API Manager redirects the user to NetIQ Access Manager, using the OAuth/OpenID Connect protocol <callout or link to OpenID connect info>, where the user is authenticated. Once authenticated, a token is returned to the client application and can be used for all future service calls.

Security for our application is centrally enforced by NetIQ Secure API Manager. When the front-end client loads, it makes service calls that have been configured to require authentication. Secure API Manager redirects the user to NetIQ Access Manager, using the OAuth/OpenID Connect protocol <callout or link to OpenID connect info>, where the user is authenticated. Once authenticated, a token is returned to the client application and can be used for all future service calls. How the user is authenticated is controlled by policies configured within NetIQ Access Manager. We could use a simple user ID and password, multi-factor authentication, or even federated authentication. The authentication method can also be made dynamic, using the Risk-Based Authentication capabilities of NetIQ Access Manager. This integration with NetIQ Access Manager is a key benefit to using NetIQ Secure API Manager.

There is no code required in the front-end client for user authentication, and neither the front-end client nor the API Manager need ever have access to the user's credentials. All the complexity of user authentication and account management is centralized to NetIQ Access Manager. This modular design simplifies application implementation, enhances security, eliminates redundant silos of authentication, and drastically improves maintainability.

Now that we know how users are authenticated, let's look at how authorization can be enforced. If we want only call center supervisors to be able to send SMS messages, we can configure NetIQ Secure API Manager to require a specific OAuth scope.

The roles and scopes are defined in NetIQ Access Manager and then read by NetIQ Secure API Manager. The NetIQ Secure API Manager administrator can then map each scope, from which we can then configure a NetIQ Access Manager role policy to provide this scope. For example, the policy could return a scope of “AllowSMS” when the user’s title is “Call Center Supervisor.” This scope would be communicated to NetIQ Secure API Manager during authentication and only users who possess that scope would be allowed to make the service call to Twilio, which would send the SMS message. NetIQ Access Manager policies are extremely powerful and flexible, making authorization within our application flexible as well. NetIQ Secure API Manager takes full advantage of the powerful authorization policy engine of NetIQ Access Manager.

NOTE: If you have multiple API endpoints, you can use one scope to control access to all of the API endpoints. You would create a different scope for an API endpoint if you wanted a different set of users to be able to access a specific API endpoint. Otherwise, it is a one-to-one relationship between the API and the NetIQ Access Manager scope.

If you have multiple API endpoints, you can use one scope to control access to all of the API endpoints. You would create a different scope for an API endpoint if you wanted a different set of users to be able to access a specific API endpoint. Otherwise, it is a one-to-one relationship between the API and the NetIQ Access Manager scope.

The flowchart below shows how NetIQ Secure API Manager evaluates authorization based on the token scope values and the user’s NetIQ Access Manager roles. The user will only get access if the needed scope was requested during authentication, was received in the token, and then only if the user has the role associated to the scope.

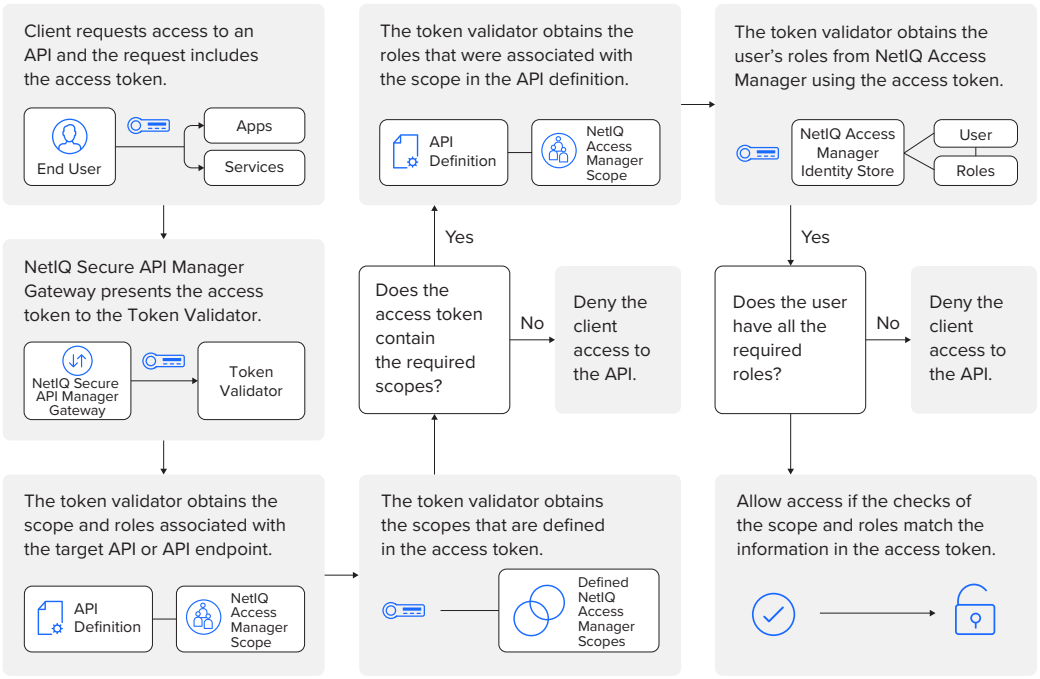

Figure 6. Token scope

The back-end service call to Twilio to send an SMS message requires authentication. Doing this authentication from the front-end client application directly would require the client code to have access to credentials for the organization's Twilio service account. NetIQ Secure API Manager offers us a better solution. The service call from the client to the NetIQ Secure API Manager gateway is authenticated using OAuth/OpenID Connect to NetIQ Access Manager, as described above. Then NetIQ Secure API Manager makes the back-end service call to Twilio using the Twilio credentials that are configured only on the gateway. NetIQ Secure API Manager transforms the request on-the-fly to provide the correct credentials to the external service. A similar process is used for calling each of the other back-end services, as each one might use a different authentication protocol or credentials. The ability to centrally and securely manage this complexity is a key benefit of NetIQ Secure API Manager.

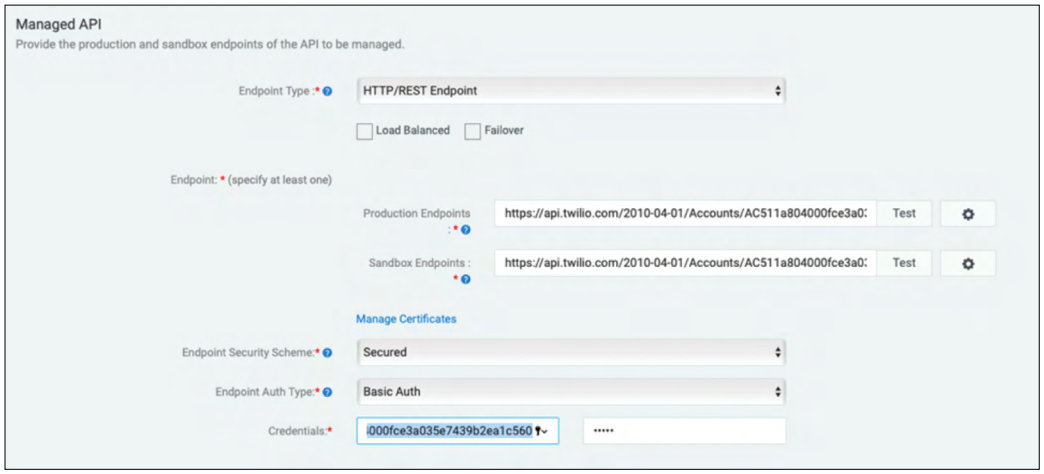

Figure 7. Adding NetIQ Secure API Manager protection to secure an API

Another benefit to being able to transform a service request at the NetIQ Secure API Manager gateway is that differences in the implementation technology of back-end services can be hidden from the client. If, for example, the back-end service is implemented as a SOAP service, NetIQ Secure API Manager can transform it into a REST service for the front-end client. Now, the client has no need to have SOAP capability of its own. Once again, handling this complexity using NetIQ Secure API Manager has simplified the implementation of the front-end client and accelerated application development.

NOTE: In most cases, you can use either REST or SOAP to achieve the same outcome (and both are infinitely scalable), with some differences in how you would configure it. REST is the most common way organizations expose their API to the public, because SOAP has components of application logic in addition to the data. Converting your SOAP calls to a REST interface through NetIQ Secure API Manager increases your protection against attacks.

In most cases, you can use either REST or SOAP to achieve the same outcome (and both are infinitely scalable), with some differences in how you would configure it. REST is the most common way organizations expose their API to the public, because SOAP has components of application logic in addition to the data. Converting your SOAP calls to a REST interface through NetIQ Secure API Manager increases your protection against attacks.

NetIQ Secure API Manager can limit the rate at which applications can call services. The rate can be limited per application and a total transaction volume per API can also be enforced. This can be used to prevent resources from being overutilized or it can be used to offer differentiated levels of service. NetIQ Secure API Manager can also be configured to control the flow of service calls to the back-end services. This can be used to smooth out utilization so that back-end services are not overwhelmed and users experience smooth performance. Caching of information from back-end services is also possible if the data in the response is somewhat static.

NetIQ Secure API Manager can also be configured to control the flow of service calls to the back-end services. This can be used to smooth out utilization so that back-end services are not overwhelmed and users experience smooth performance.

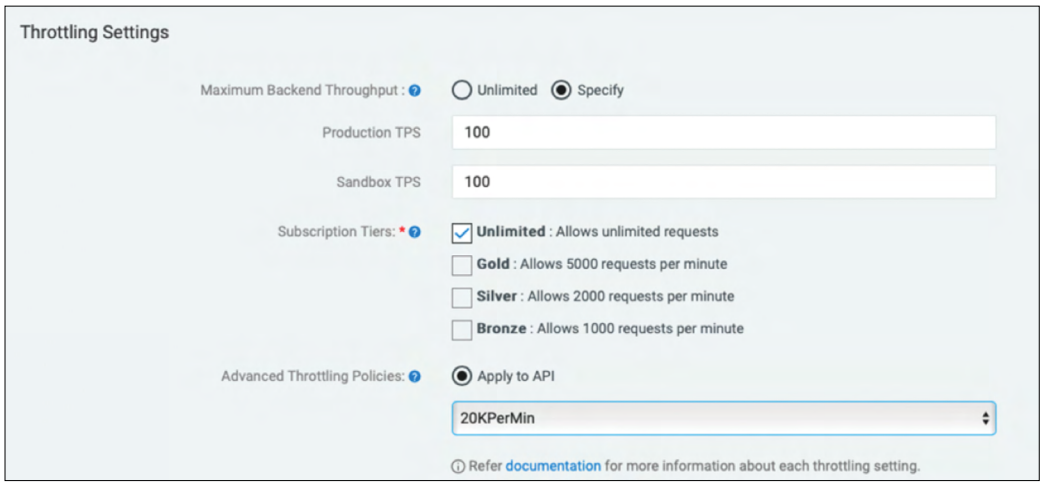

Figure 8. Throttle API access to match your environment design

Building Out Our Test Environment

NetIQ Secure API Manager can be deployed as a single appliance for testing and development. However, for production deployment, it is recommended that NetIQ Secure API Manager components be deployed on separate appliances. At a minimum, the Database Service should be deployed separately. An enterprise deployment where each component is deployed separately is shown in the diagram on the following page In this example, even the analytics component has its own appliance instance.

Such a deployment makes sense for a highly utilized system, but many deployments do not require that level of resources. For the example application, we are going to assume that the number of concurrent sessions will not exceed 400 and that the request rate will be less than 400 requests per second.

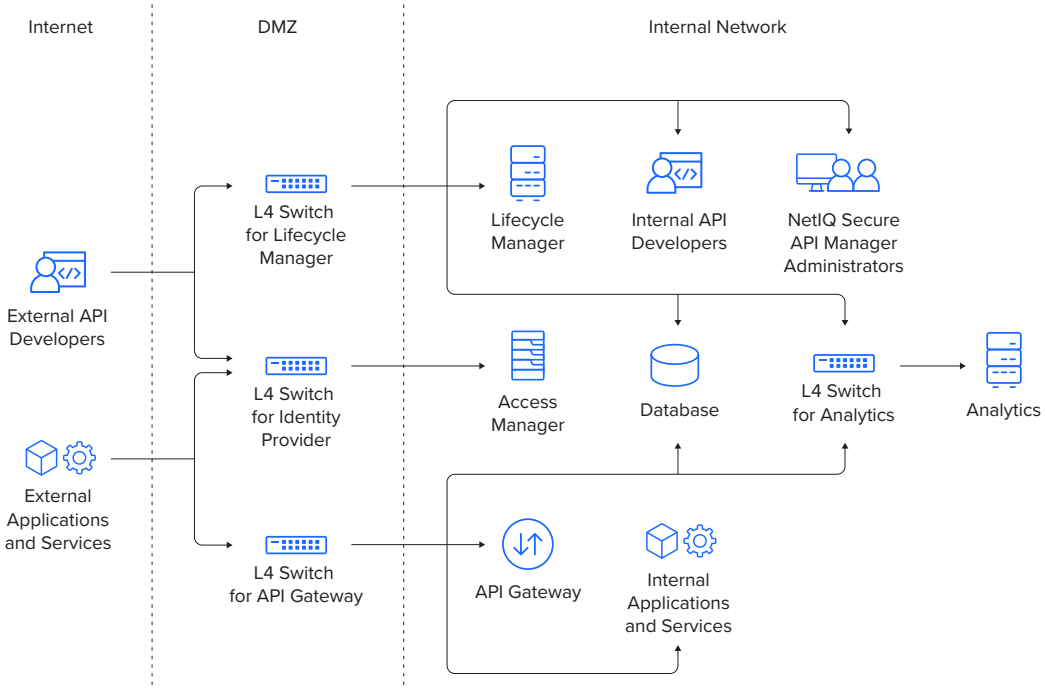

Figure 9. Components of a deployment

There is a requirement for the application to be highly available, so we will implement load balancing to multiple appliances, each running the Lifecycle Manager and the Analytics components. These components support clustering and failover.

It is still a good idea to run the Database Service on its own appliance, so we will do that in our example. This means that we will need a minimum of two appliances before we build any redundancy into the system. There is a requirement for the application to be highly available, so we will implement load balancing to multiple appliances, each running the Lifecycle Manager and the Analytics components. These components support clustering and failover. Cluster implementation requires that the load balancers be configured with session persistence so that requests from the same client are routed to the same node, unless that becomes unavailable. The nodes must also share a Network File System (NFS) mount so that common deployed content and configuration is available to all nodes. We will use a highly available NFS appliance to provide fault tolerance for the mounted file system. The Database Service does not support clustering, so we will use VMware High Availability to ensure that the service is always available. We will also periodically back up the database so that it can be easily restored in case of disaster.

The diagram below shows the deployed appliance and network configuration. One goal of the selected design was to minimize the number of appliances required. Additional nodes could be deployed to support high loads and we could choose to further segregate the components if needed. The selected design should provide sufficient performance and security for most implementations. We have chosen to illustrate an enterprise-based deployment, but a cloud deployment is also possible. Please contact OpenText™ for information about cloud deployment.

Another key benefit of using the NetIQ Secure API Manager gateway is that it provides an easy way to centrally collect information about API usage and performance.

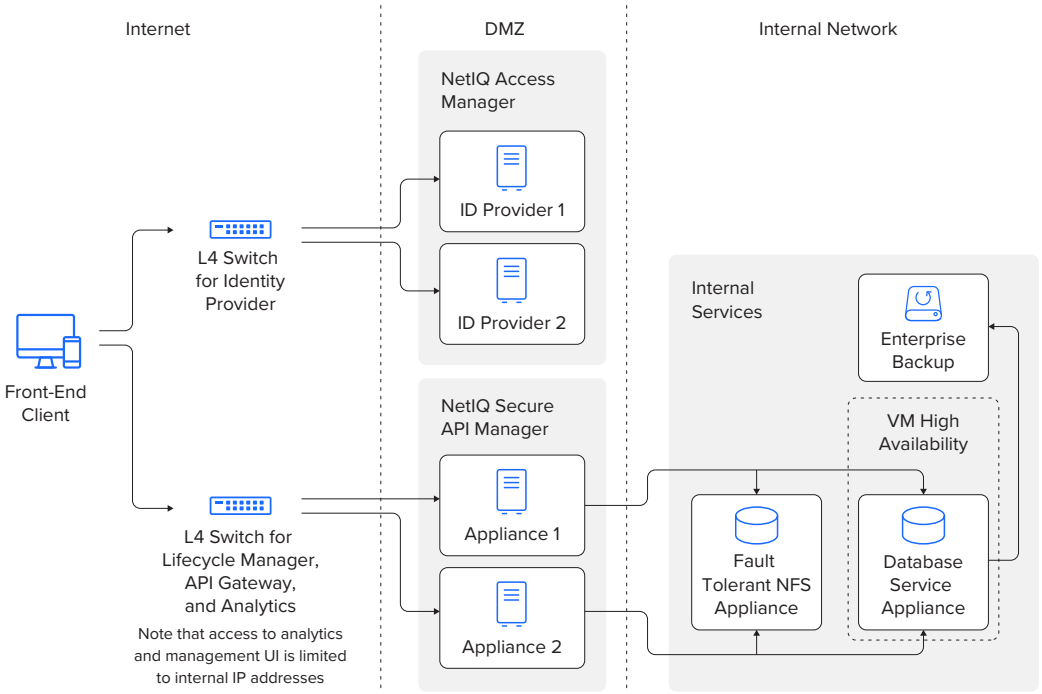

Figure 10. Boilerplate network configuration

Analytics and Monitoring

Another key benefit of using the NetIQ Secure API Manager gateway is that it provides an easy way to centrally collect information about API usage and performance. NetIQ Secure API Manager provides a number of built-in analytics functions to present this information to both administrators and line-of-business users.

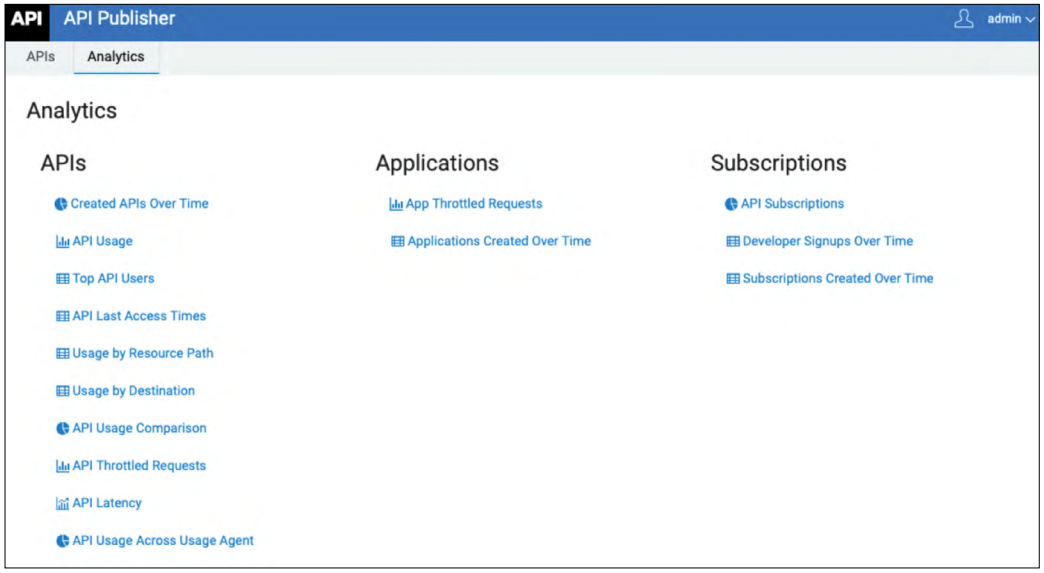

Figure 11.

NetIQ Secure API Manager provides a number of built-in analytics functions to present this information to both administrators and line-of-business users.

Of course, the raw data is also available so that you can implement your own analytics, if needed. The example below shows a graph of API utilization, enabling you to easily track which APIs are consuming system resources.

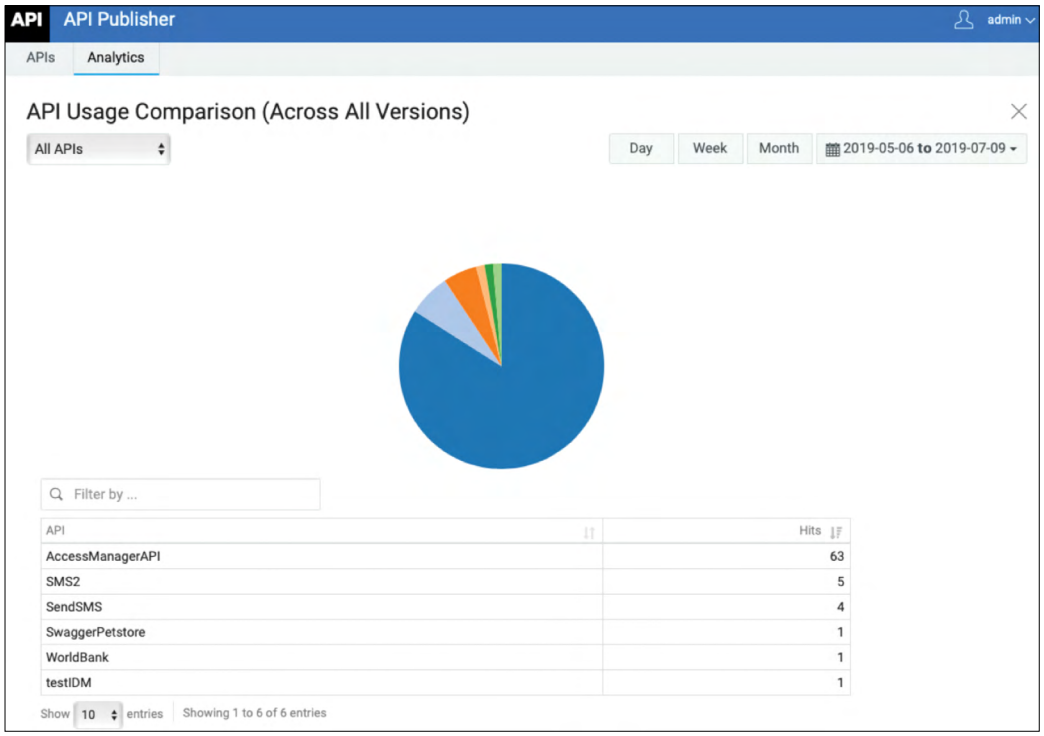

Figure 12.

Tthe raw data is also available so that you can implement your own analytics, if needed.

NetIQ Secure API Manager also collects system performance data, enabling you to effectively monitor and manage your system. Detailed information is captured about system resource utilization and API performance. The example below shows a summary view of the system performance.

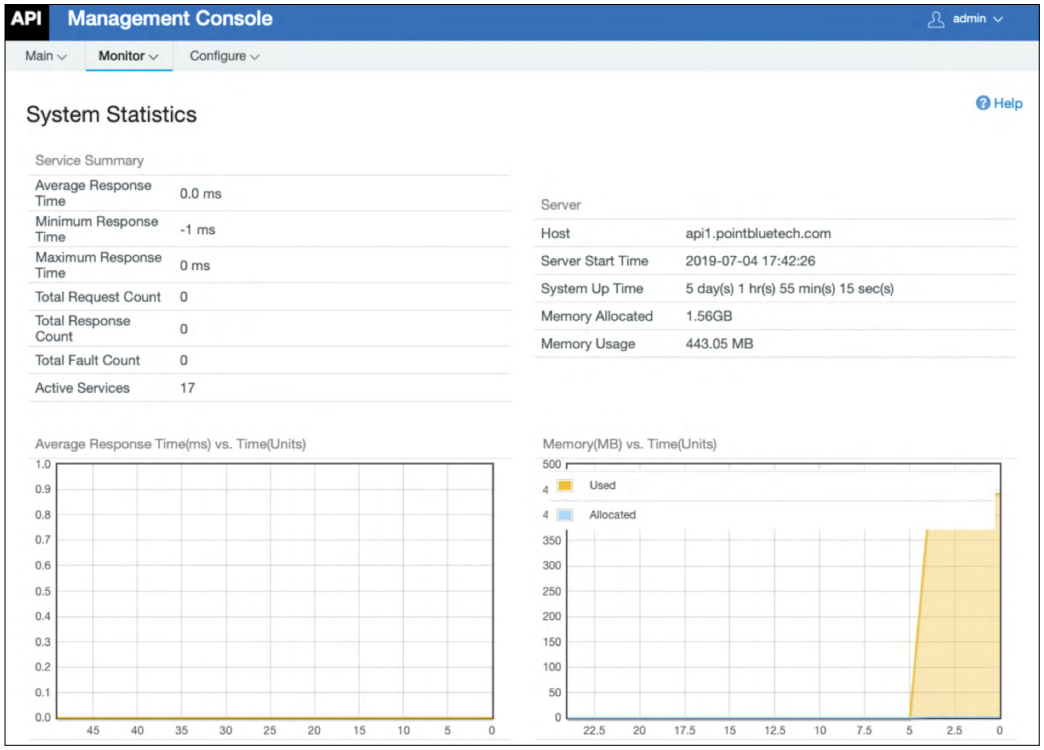

Figure 13.

NetIQ Secure API Manager also collects system performance data, enabling you to effectively monitor and manage your system. Detailed information is captured about system resource utilization and API performance.

Conclusion

NetIQ Secure API Manager extends your access and authentication environment to include secure API delivery for all your secure integration needs. As organizations increasingly look for new ways to leverage their digital assets to expand their business into more efficient models, this simple example application has demonstrated how you can secure your microservices-based applications. NetIQ Secure API Manager provides the control and central point of administration needed to provide API manageability and improved implementation time. It also offers a higher level of API security compared to traditional hardening practices. Its seamless integration with NetIQ Access Manager makes NetIQ Secure API Manager the obvious choice for existing customers to extend their access management architecture.

NetIQ Secure API Manager provides the control and central point of administration needed to provide API manage-ability and improved implementation time

About NetIQ by OpenText

OpenText has completed the purchase of Micro Focus, including CyberRes. Our combined expertise expands our security offerings to help customers protect sensitive information by automating privilege and access control to ensure appropriate access to applications, data, and resources. NetIQ Identity and Access Management is part of OpenText Cybersecurity, which provides comprehensive security solutions for companies and partners of all sizes.

Connect with Us

www.opentext.com

opentext™ | Cybersecurity

OpenText Cybersecurity provides comprehensive security solutions for companies and partners of all sizes. From prevention, detection and response to recovery, investigation and compliance, our unified end-to-end platform helps customers build cyber resilience via a holistic security portfolio. Powered by actionable insights from our real-time and contextual threat intelligence, OpenText Cybersecurity customers benefit from high efficacy products, a compliant experience and simplified security to help manage business risk.