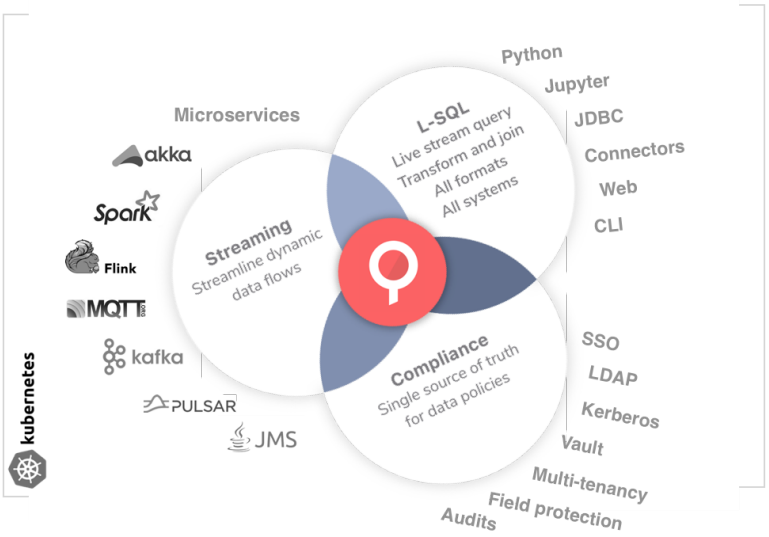
LENSES

LENSES = f(data)

The catalyst for open functional data & flows.



Lenses Reference Architecture

kafka kubernetes

Andrew Stevenson, CTO
Spiros Economakis, DevOps

All rights reserved. Lenses is trademark and registered trademarks of Landoop Ltd. in the United Kingdom and other countries. All other brand names, product names, or trademarks belong to their respective owners.

Prerequisites

This document does not provide details configuration or installation setup of Lenses ®. Please refer to the full https://lenses.io/docs for this.

In addition this document does not describe the installation or best practices for the underlying middleware such as Apache Kafka®.

Introduction

This reference architecture establishes the best practices for running Lenses either on physical machines, in cloud providers or Kubernetes with Apache Kafka® as the middleware.

Lenses, a streaming focused DataOps platform, provides cutting edge visibility into data streams, integration, processing, operations and governance for the modern data-driven enterprise.

DataOps

Big Data, fast data, IoT data, small data. Data usage is growing, organizations are transitioning to become data-driven. Data analytics and integrations are becoming more critical and data needs to be shared and made available to the whole enterprise.

Software development teams, organised around DevOps principles, are able to deliver high quality applications to production via automated and repeatable methods. Creating cross functional teams of developers and operations, working with one culture allows this fast, iterative approach.

DataOps builds on the lessons of DevOps to allow data engineers or any data literate user to build and operate production ready data streams and deploy quickly. DataOps follows, at its core, the same principles as DevOps:

1. Integration, Processing and Analytics as code
2. Config is code
3. Automate as much as possible
4. Make it repeatable
5. Think of data flows not servers.

DataOps puts the data engineers and analysts at the heart of the process. Data engineering has been and will continue to play an important role in any organisation. Typically an organisation

will already have a large data engineering capacity for ongoing integration, data marts and data warehouses.

Moreover, as companies look towards Big Data solutions, focus shifts to software developers. However, this can lead to lost opportunities and time to value. Existing data literate teams often struggle to pick up the new technology and their knowledge of the organisations data is **not** being utilised. DataOps, focuses on the data, allowing all parties in an organisation the opportunity to innovate.

DevOps to DataOps

We appreciate and understand data processing and engineering. If we analyze DevOps, it is the lifecycle from Development → Operations.

Development means the **software development life cycle** that is: design, implement, unit and integration test, package and document it. Operations means deployment, promoting across environments, monitoring and alerting.

DevOps activity usually spans across multiple sprints, and is an organized but relative slow way for bringing business intelligent data applications into production. DataOps speeds the ***-Dev-*** aspects of the lifecycle and introduces a self-service way of operating over data, for all.

Lenses is fully automating the deployment of SQL continuous queries, processors and connectors in a scalable and fault-tolerant way. Not only that, but it can automate all the operational aspects of the data in motion. Monitoring and alerting out of the box.

The end result? With DataOps you can move applications to production in minutes, instead of weeks. Lenses makes it easy and simple to bring a DataOps to your data-driven organisation, while at the same time integrating with all existing Applications, micro-services and data pipelines.

Data Visibility

Data is available from many sources in all organisations. Opening up this data to members of an organisation can be a challenge. Data integration projects aim to provide a unified domain model but require developer skill sets to provide insight.

Lenses provides visibility into your data via SQL. This is possible via the user interface or via rest and websocket endpoints exposed by Lenses. SQL capability opens access to entire organization, allowing them to query data continuously or historically. For example business users can filter data flowing through the system for a GDPR related enquiry or a risk manager in a trading firm can query, aggregate and group the live trades.

Data Transformation

Data needs to be filtered, joined, cleaned, aggregated. Lenses opens data visibility via SQL and does the same for data transformations.

Data transformation is made easy and simple with Lenses. Click and deploy, manage and monitor the deployment of Continuous SQL processors to join, aggregate, filter and enrich streaming data.

Development cycles are reduced and applications move into production faster. Importantly you can reuse existing skills rather than battle to hire and retain specialist developers. Continuous SQL processors can be easily scaled out using Apache Kafka® Connect or Kubernetes. Lenses handles this for users providing a consistent and transparent experience.

Data Security and Policies

Security is a first class citizen in Lenses. It builds on the authorisation, authentication and encryption of underlying middleware by providing role based access. The role based security model allows for fine grain permissions to be set for users or LDAP groups. This enables administrators to limit functionality and restrict viewing of data via whitelists, blacklists and namespaces.

This `topic-centric` security model and usage of namespaces enables a high-level of multi-tenancy. It also extends out automatically, so that if a user has been granted read-access for example to a particular topic, he automatically has read-access to all applications that are using that data, so that he can understand its usage (consumers, producers, connectors, processors, microservices, pipelines).

The `field-centric` security model, complements the topic-centric access level security. According to the *"Guide to Protecting the Confidentiality of Personally Identifiable Information (PII) from NIST (National Institute of Standards and Technology)"*[1], organisations should i) identify and ii) manage all PII residing in their environments.

[1] https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=904990

Lenses continuously collects and maintains metadata and knowledge about the underlying data both in the middleware and other data systems. In addition to querying and identifying data, with the data officer role, you can apply a data policy. Policies are automatically applied across your data platform applying redaction for PII where required.

Going beyond authorization, authentication, topic and field centric security, all actions are `audited` and stored in an immutable way including all queries. Now you can answer ***"who did what and when"***. This provides enterprise level security and a presentation layer with full auditing to comply with modern data regulations.

Monitoring

Lenses includes monitoring of the underlying middleware, including Apache Kafka® clusters, and provides integration with Prometheus and AlertManager, two popular open source monitoring projects from the Cloud Native Computing Foundation.

Lenses enriches and integrates with the best of breed monitoring systems with critical additional information about the middleware layer, complementing and completing your platform monitoring solution and intelligently sending out notifications.

We recognize operators want a reduced set of tools and centralized alerting. Middleware is a vital part of your real time streaming platform but not the only component. Lenses provides detailed domain specific monitoring knowledge to complement Prometheus and Grafana.

Eco-system

Lenses at its very core is: SQL plus APIs. There are REST or WebSocket APIs for everything. To make life easier for data operators a rich set of open-source clients are available:

1. Python
2. Go
3. JDBC
4. Javascript Redux/React

The clients enable developers and operators to build, deploy and monitor application topologies and data scientists to access data in motion.

For example, a `Python` developer can deploy connectors to source data, SQL processors to transform data and subscribe and consume data all from his Jupyter notebook for further analysis .

A `DevOps` can use the command line interface (CLI) that is built using the open source Go client. He can build, deploy and promote flows to production with confidence and automate flows via version control following a DataOps approach.

A `BI` application or even an Apache Spark or Apache Flink project, can use the JDBC driver to consume or produce data. The ODBC driver that is planned for public release soon, will open further access to a wider Business Intelligence audience.

A `FrontEnd` engineer can aggregate data, and then using the JavaScript open source library available for React (that is using the Redux middleware) and build a real-time dashboard and graph, like a feed for financial trading usage.



More information about Lenses clients at github.com/lenses

Streaming Ecosystem

There are numerous popular Streaming and Data processing technologies in usage today. From Apache Spark, Apache Flink, Akka Streams and KStreams and to the vast numbers of custom micro-services, that are consuming and producing data.

All of them can be displayed on the topology screen, so that everyone can easily identify all the data pipelines and the entire application landscape. Monitoring, health-checks and alerts can now be applied at every layer of the data journey. This centralized view of the world, irrespective of the underlying technology and data format can provide insights and answer questions on how my data is utilized.

In combination with the rich metadata, you can now identify how and where PII or other sensitive data are processed.

Lenses is a place for architects, data engineers, data officers and business analysts to better understand and interact with their data.

Deploying Lenses

The following section will focus on how to deploy Lenses. Lenses is built with machine sympathy, and should be installed near your middleware (i.e. in the same datacenter). Apart from that, it is a JVM application, easy to deploy and run. Under the hood multiple producers and consumers are running and inspecting the underlying data as well as the infrastructure.

Storage

Lenses requires a minimum of 500 MB for storage of it's configuration files and log files. All other state is persisted in middleware layer recovered at startup. Those topics are automatically managed.

Hardware Sizes

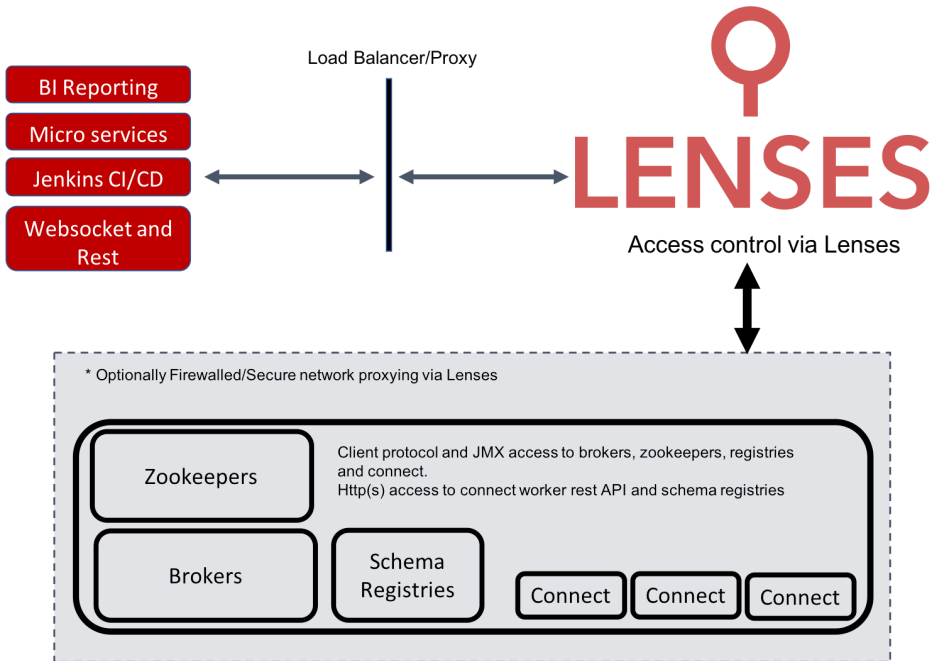
Hardware sizing can depend on a number of factors:

1. How many brokers
2. How many partitions in your topics
3. How many concurrent users you expect

Typically a 4 CPU instance with 8GB RAM is more than sufficient to manage medium sized infrastructure (i.e. up to 12 x brokers). For larger-scale installations such as 20 x brokers and opening up access to 400+ Data Engineers, we would typically recommend a 6 CPU instance with 12GB RAM.

Location

Lenses should be deployed on a server as close as possible to your Apache Kafka® cluster, we do not recommend co-locating with the brokers. Lenses can be treated like any other client of Apache Kafka®.



Connectivity

Apache Kafka® Cluster

Lenses runs producers and consumers, additionally it can perform administrative tasks such as topic, acl, quotas and schema management and therefore connectivity is required to Apache Kafka® brokers.

For the best experience we also recommend connectivity to

1. Zookeepers
2. Schema registries *
3. Connect clusters

We recommend securing the network layer of the core cluster services. Lenses can act as proxy to control client access and deploy Connectors, including SQL Processors.

* *Lenses works with both Confluents and Hortonworks schema registries.*

Lenses Clients

Clients of Lenses, such as JDBC applications, Go microservices, Frontend applications or your CI/CD pipelines require HTTP(s) access to Lenses. Lenses will act as a proxy to control data access to the core cluster services and provide governance and auditing.

JMX

Apache Kafka and its components all expose metrics via JMX. To utilize the full features of Lenses JMX should be enabled on the Apache Kafka brokers, Zookeepers, Kafka Connect clusters and any Schema Registries. Details on how to enable JMX can be found in the Lenses documentation.

TLS termination

Lenses currently does not offer TLS support. To secure communication, Lenses should be run behind a proxy such as NGINX.

Continuous SQL processors and connectors

Landoop provides Kafka Connect Connectors for nearly all the major data system to stream data in and out of Kafka. Our connectors also support SQL for simplified configuration but we also support Lenses SQL Processors running in Kafka Connect. This allows you to scale out processing using vanilla Apache Kafka distributions.

Lenses allows you to push SQL processing out to Apache Kafka Connect. Processors can then be deployed as normal via Lenses. Note that while it's possible to manage and deploy by the Connectors section of Lenses we recommend its is done via the Processors interface to provide a richer experience.

To enable SQL processing in Connect the Lenses SQL Processor connector must be made available to each workers. We recommend using the `*plugin.path*` option to achieve classloader isolation. Set this option in the worker properties file for each connect worker.

1 Cluster vs Many

Connect has two deployment modes, standalone and distributed. The latter is the intended deployment mode for production. In this mode workers form a cluster and new connectors can be submitted and managed via a rest api. Each cluster can run any connector that is available to it on the classpath or the `*plugin.path*` and it can run multiple instances of each.

Having a single cluster has some advantages:

1. Less servers to manage. Fewer servers to provision, maintain and distribute connectors to. Ideally each worker should be on a seperate machine for better fault tolerance.
2. Central cluster for users to go to.

However, it does have some disadvantages:

1. Adding new connector jars requires a restart of clusters which stops flows. Connect will restart for source connectors where it left of but for streaming sources source as Twitter this can result in data loss. Apache Kafka® consumer group semantics ensure sink connectors start from where they left of reducing the chance of data loss.
2. Submitting new connectors causes Connect to rebalance all running connectors. This means a shutdown of connector instance, rebalance, across the workers and a restart.
3. Removing new connectors causes Connect to rebalance all running connectors. This means a shutdown of connector instance, rebalance, across the workers and a restart.
4. Heap space issue in one instance can impact other connectors.

An better alternative is running multiple Connect cluster per business unit or team or even an individual cluster per connector. This provides better isolation between connector instances and the rebalances that occur in the Connect framework from user actions or failures.

Kubernetes

Lenses in Kubernetes

If you are utilizing Kubernetes to deploy your workloads Lenses can deploy connectors and SQL processors from either inside Kubernetes or outside. In addition, if running outside of Kubernetes Lenses can deploy and manage SQL processors across multiple Kubernetes clusters.

Lenses stores its state in Apache Kafka®. Lenses can be simply installed using a Kubernetes deployment resource. StatefulSets are not required. Lenses requires a small amount of storage space for its configuration which can be mounted via Kubernetes Secrets and configMaps.

Kubernetes Configuration

Lenses can deploy, monitor and manage SQL Processors by providing a valid kubeconfig file if installed outside a Kubernetes cluster but we recommend to install Lenses inside your cluster.

You can use the **`kubeconf`** to configure multiple Kubernetes clusters. If no Kubernetes config is provided Lenses will auto configure itself for the cluster it is hosted in.

Enterprise customers have access to the Lenses SQL Processor image, the version of the image is set in the `lenses.conf` file. This requires that the service account Lenses uses must be patched with an image pull secret to allow it access to Landoop's docker registry.

Kubernetes Resources

Lenses provides defaults for the pod request and resource limits. Applications should not be deployed into Kubernetes with no limits set. The requests and limits can be adjusted accordingly in the **`lenses.conf`** file.

The defaults currently are:

Memory request: 128M

Memory limits: 512M

Future versions will allow customising the memory and cpu requests per application deployed and managed via Lenses.

Kubernetes RBAC

For RBAC enabled Kubernetes cluster the service account used to deploy Lenses requires access to watch, deploy and modify pods, deployments and statefulsets. The following verbs are need for each:

1. list
2. watch
3. get
4. create
5. update
6. Delete

For more information see the documentation.

Lenses Docker Image

The Lenses docker image is available on DockerHub. This docker will construct the Lenses configuration files from environment variables starting with LENSES_. Sensitive data, user passwords and license keys should be mounted and sourced from Kubernetes secrets.

Lenses SQL Processor Image

The service account you use to deploy SQL processors must be patched with an image pull secret to access Landoops Docker registry.

Kubernetes Secrets

Lenses requires various secrets for user access. Lenses uses an external file ***security.conf***, separate from the main configuration that should be lockdown by admins. In Kubernetes this file should be sourced via Kubernetes secrets.

For Apache Kafka® SSL binary key and truststores need to be mounted. This should be done by providing the base64 encoded contents to the Kubernetes secret and then mounting the contents.

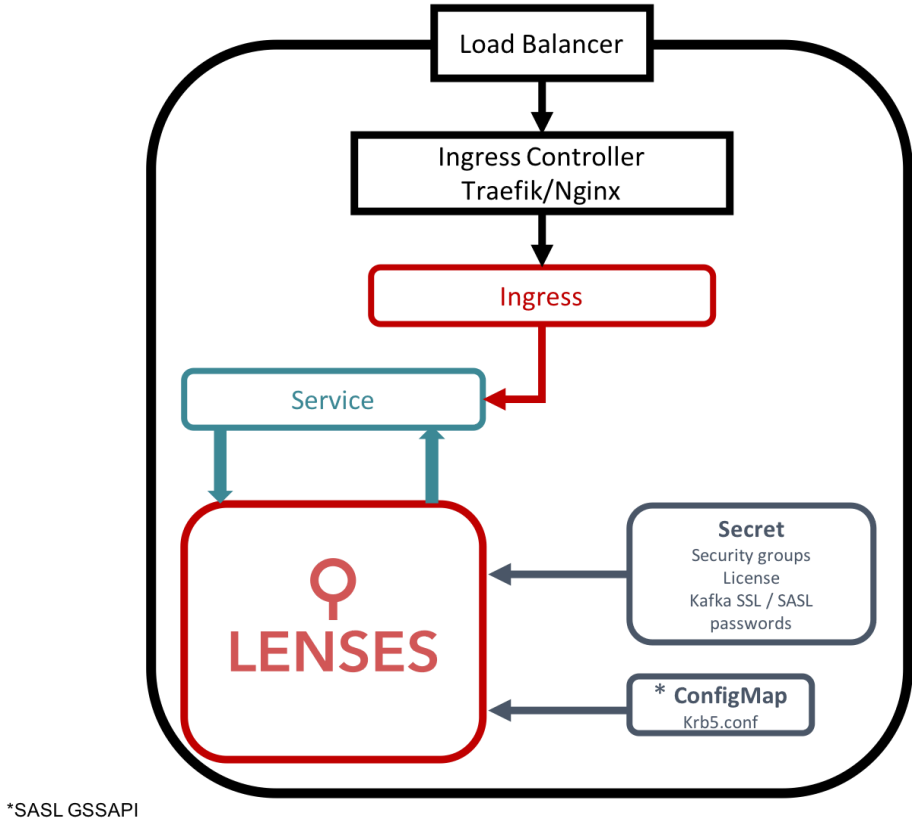
Kafka, Zookeeper, and Schema Registry services in Kubernetes

Apache Kafka clusters can be deployed inside or outside of Kubernetes, if deployed outside the client and JMX endpoints of each should be reachable and set in the Lenses configuration file. If deployed inside statefulsets should be used and Lenses configured with the headless service for the service endpoints.

Lenses Helm Charts

Helm is a package manager for Kubernetes and simplifies the install using a configuration driven approach. Helm charts are available for Lenses, Lenses SQL runners and all the Apache Kafka Connect Connectors in Lenses connector collection.

Deploying Lenses inside Kubernetes is recommended. The charts will construct the necessary secrets, confimaps, service and ingress routes required. A reverse proxy such as Traefik, Nginx, or Envoy is left up the Kubernetes administrators.



*SASL GSSAPI

Lenses SQL Processor Helm Chart

A Helm chart is available for the Lenses SQL Processors. You can use Helm to deploy you processor, Lenses will still track and monitor their status and you can manage further actions such as scaling, stopping, restarting and deleting via Lenses. Lenses tracks the deployments and pods via labels. Do not remove or adjust the labels.

Note that any modifications made to the SQL Processors from Lenses will not be reflected in Tiller, the Helm server side component.

We recommend to deploy and manage Processors via Lenses. Lenses audits all actions but information regarding “who, what, and why” can be lost if deployed outside of Lenses scope.

Lenses SQL Processor Kubernetes Manifest

If you prefer to use standard Kubernetes manifest to deploy with kubectl from outside of Lenses ensure you set the labels correctly. Lenses tracks Kubernetes resources based on labels. You must ensure the following label are set:

```
lenses.io/app: my-app-name
lenses.io/app.type: lenses-processor
```

As with deploying by Helm, we recommend to deploy and manage Processors via Lenses. Lenses audits all actions but information regarding “who, what, and why” is lost if deployed outside of Lenses scope. However, performing actions on the processors from Lenses will not result in Helm Tiller drift from the actual state.

Deployment Scenarios

Kubernetes offers great flexibility as such there’s three main scenarios when deploying Lenses and working with Kubernetes:

1. Lenses and your Apache Kafka® cluster are outside Kubernetes with SQL processing scaling inside
2. Lenses is inside Kubernetes and Apache Kafka® clusters are outside
3. Lenses and Apache Kafka® clusters are inside Kubernetes.

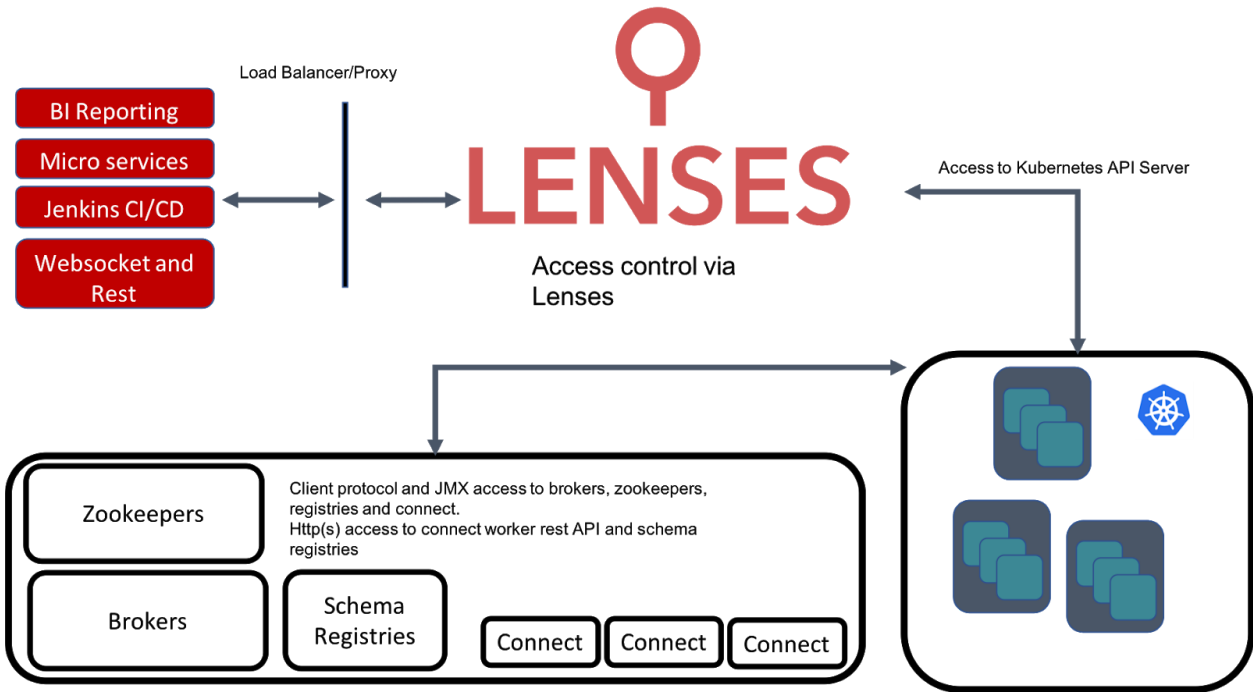
Lenses and Apache Kafka® outside Kubernetes and SQL processors inside

This scenario is typical for organisations with existing Apache Kafka® clusters migrating workloads into Kubernetes. Apache Kafka® clusters have been installed on premise or in cloud providers and application workloads such as Lenses SQL processors are deployed into Kubernetes.

Lenses can simply be configured to use to a valid Kubernetes “kubectl” config file to deploy and monitor SQL processors in Kubernetes and access the Apache Kafka cluster services deployed outside.

Connectivity from Kubernetes to Apache Kafka®

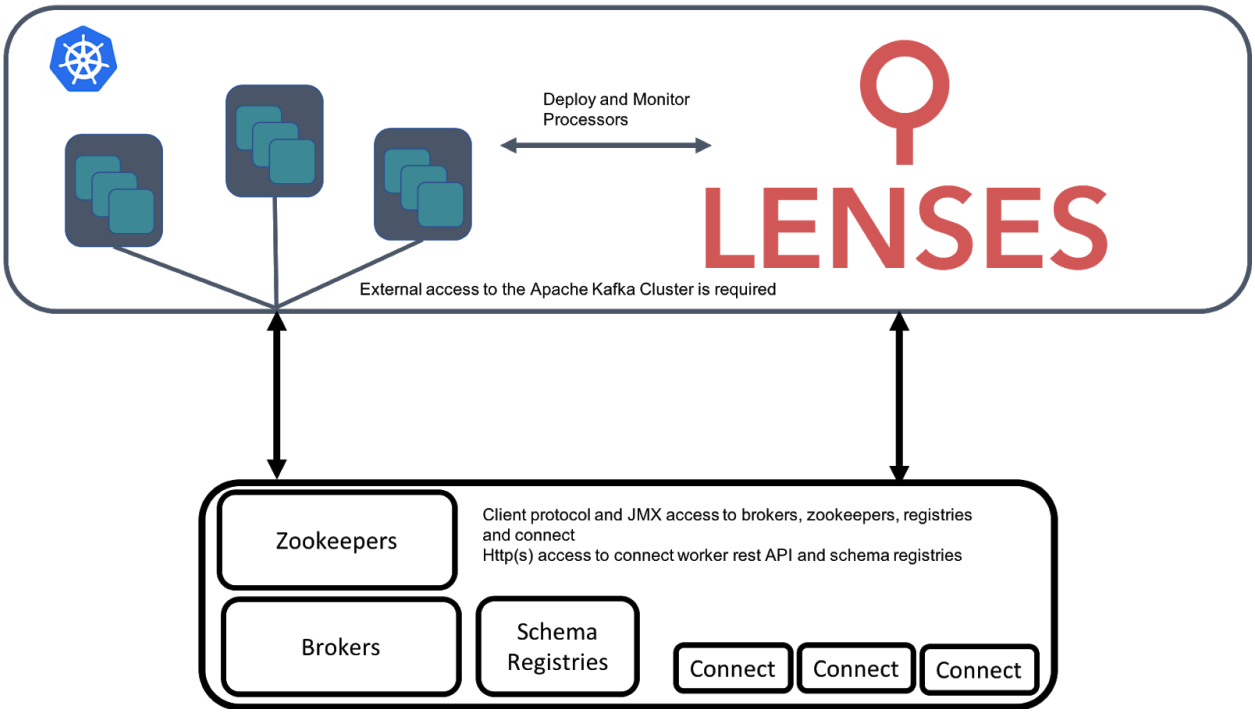
Lenses automatically configures SQL processors with information such as the IP Address and host names of the brokers, schema registries, zookeepers and connect workers. These services need to be reachable from the pods, for example, in Azure, access is required from the Kubernetes cluster VNET to the Apache Kafka® cluster VNET.



Lenses inside Kubernetes - Apache Kafka® outside

If your are running Kubernetes and have an existing Apache Kafka® cluster deployed on premise or with a cloud provider it is often better to deploy Lenses into Kubernetes. This can easily be achieved and automated via our Helm charts as previously discussed.

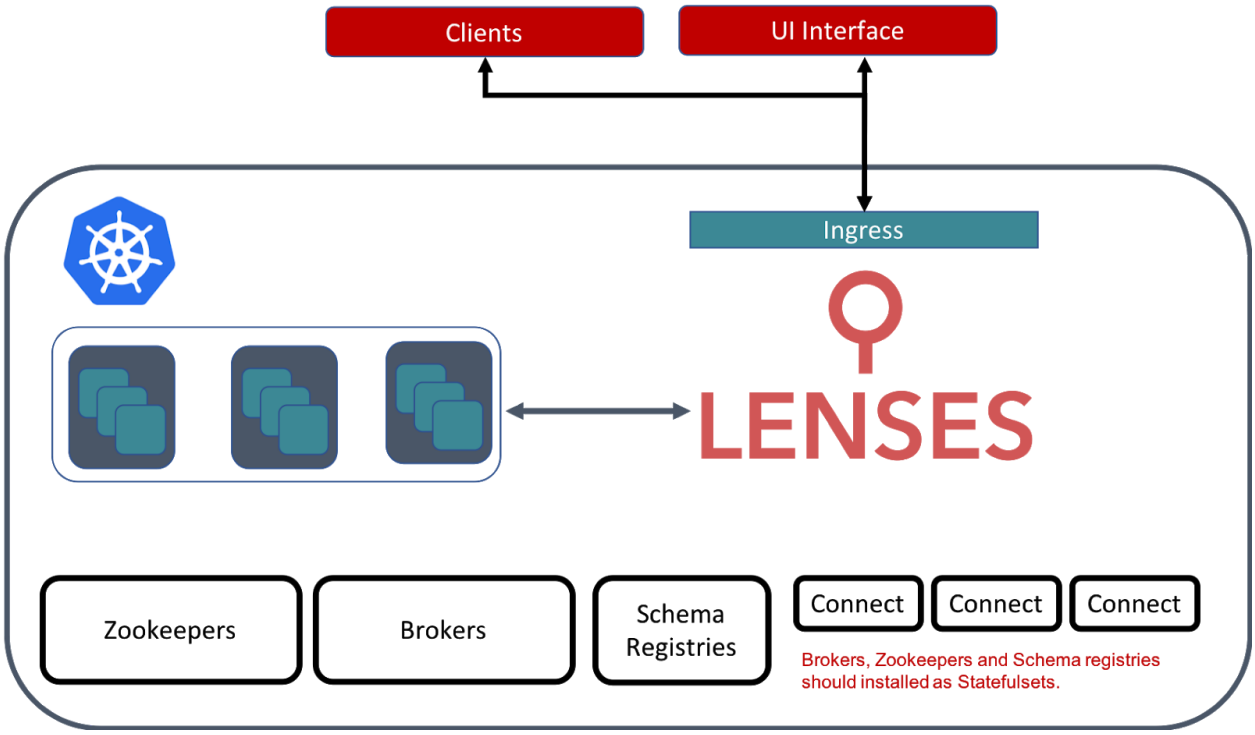
In this scenario again the Kubernetes cluster nodes will require access to the Apache Kafka® cluster services.



Lenses and Apache Kafka® inside Kubernetes

Lenses stores all its state inside Apache Kafka® and recovers application landscapes from Kubernetes so is a good fit to run inside. Apache Kafka® and the supporting components, however have state, potentially large state that needs to managed. Several Helm charts are available to install a Apache Kafka® cluster. We recommend installing these services as statefulsets to give ordinal index identifiers, stable persistence and stable network identifiers. Statefulsets provide headless services, Lenses must be configured for each pods' network identifier.

If you have deployed your Apache Kafka® cluster inside Kubernetes we recommend you also deploy Lenses into Kubernetes.



Connect in Kubernetes

Multiple connect cluster should be created with statefulsets and exposed via head services to Lenses to reduce the impact of rebalances on connectors. Using statefulsets for the deployment has the additional benefit of providing a stable network identifier which can be supplied to Lenses so it can address the rest endpoints of the workers.

In Kubernetes you can create one cluster per connector instance. Kubernetes makes this easy. Each cluster is a Kubernetes statefulset or deployment resource with the number of pods reflecting the `tasks.max`. The Stream Reactor projects has Helm charts which create statefulset deployments with one cluster per instance. If this approach is taken the Lenses configuration file needs to be updated for Lenses to manage the connectors.

AWS

Lenses in AWS

If you are using AWS you can easily deploy Lenses and create your own cluster. Lenses can be deployed in AWS and pre-configure it with AWS MSK, your current Apache Kafka® or Lenses can autodiscover your Apache Kafka® cluster given the proper AWS IAM permissions and having the proper AWS resources tagging.

In the cloud, security is really important and because Lenses access sensitive data it should not be exposed directly to the public Internet but there are a couple of secure ways to deploy Lenses in Cloud.

You can find Lenses available in AWS Marketplace here

AWS Resources

EC2

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.

Lenses can run in an EC2 instance using our own CloudFormation template which is available in the AWS Marketplace.

ECS

Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container orchestration service that supports Docker containers and allows you to easily run and scale containerized applications on AWS. Amazon ECS eliminates the need for you to install and operate your own container orchestration software, manage and scale a cluster of virtual machines, or schedule containers on those virtual machines.

Lenses can run in an ECS as a cluster using the docker image which is available on DockerHub or using our own CloudFormation template.

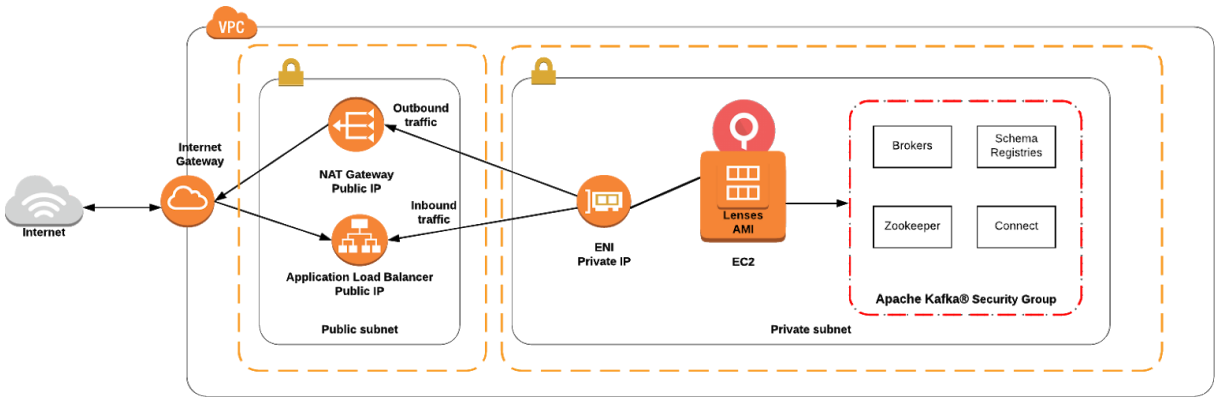
Deployment Scenarios

AWS offers great flexibility as such there's three main scenarios when deploying Lenses:

- Lenses as an EC2 instance and Apache Kafka® in same VPC
- Lenses as an EC2 instance and Apache Kafka® in different VPC
- Lenses in ECS with your own Apache Kafka® in same VPC
- Lenses in ECS with your own Apache Kafka® in different VPC

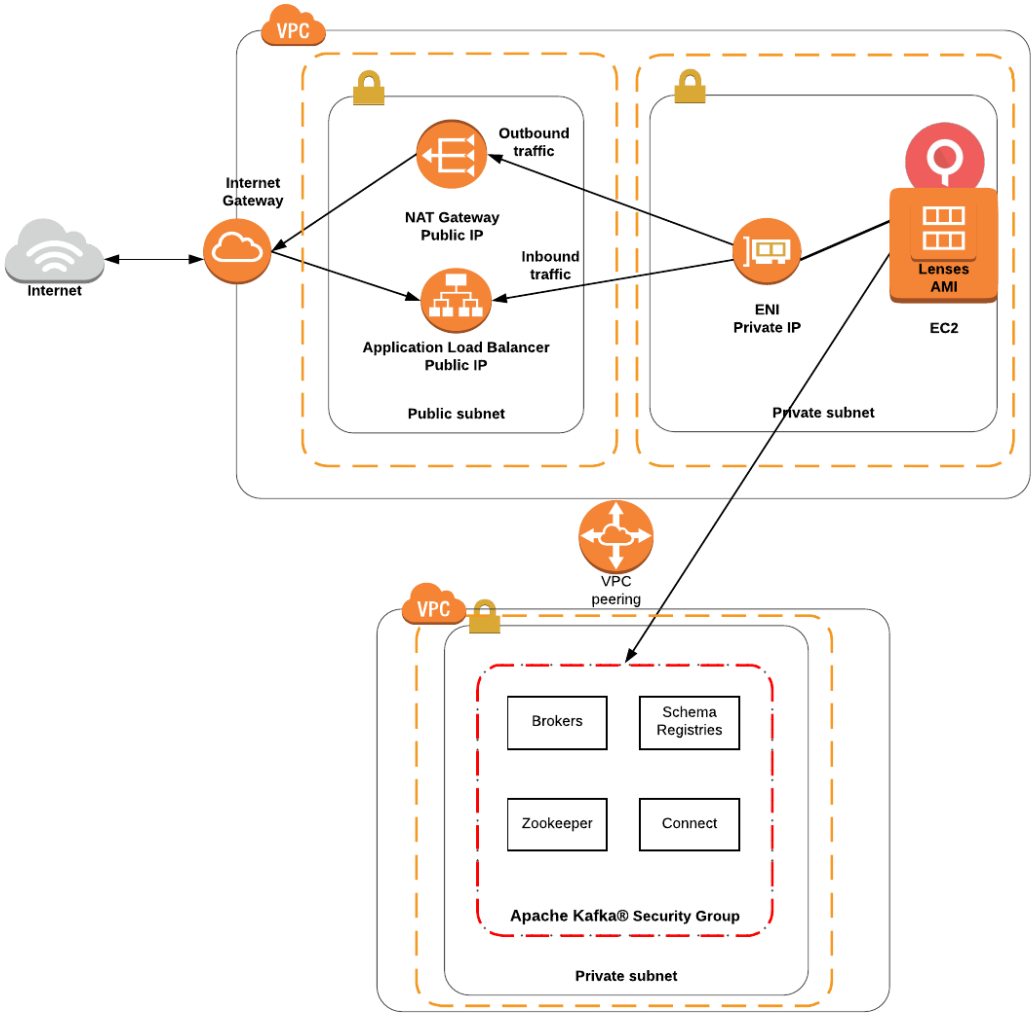
Lenses in EC2 and Apache Kafka® in same VPC

Now we have again a network setup where Lenses is deployed to the same VPC and same private subnet but as an EC2 instance which uses our own Amazon Machine Image (AMI). In the EC2 instance, a persistent disk is attached in order to keep the data for Data Policies module.



Lenses in EC2 and Apache Kafka® in different VPC

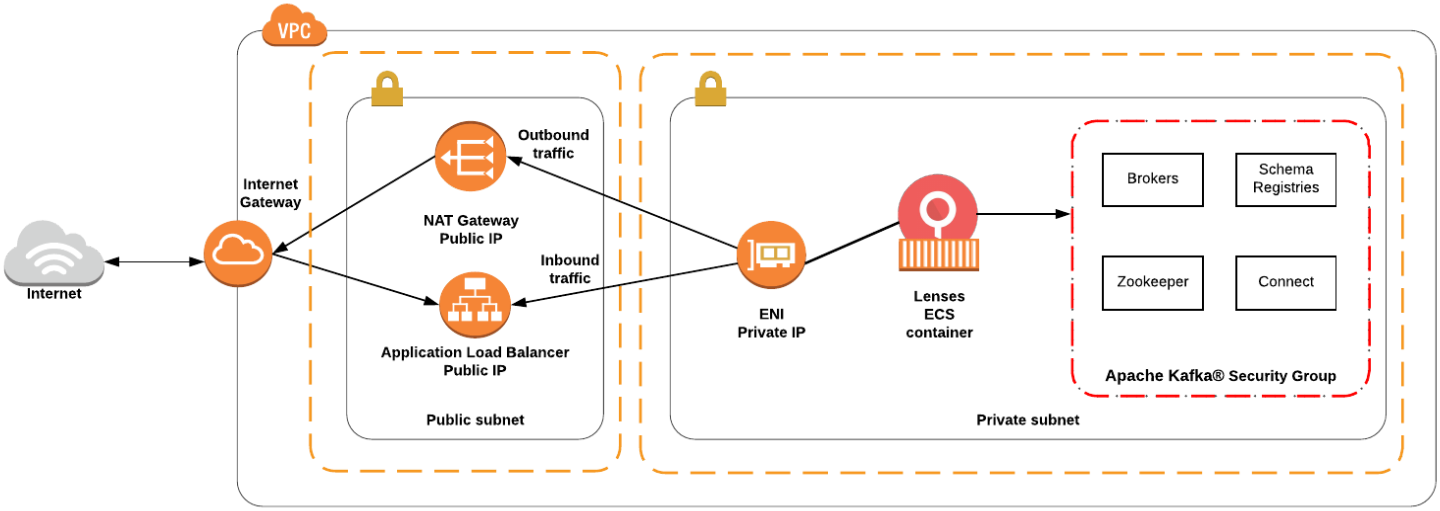
The only difference from the previous one is that right now there are two separate VPCs which are connected with VPC peering in order to allow Lenses access Apache Kafka® in the different VPC. Again the Security Group to allow Ingress traffic is needed. In the EC2 instance, a persistent disk is attached in order to keep the data for Data Policies module.



Lenses in ECS and Apache Kafka® in same VPC

This architecture deploys Lenses in an ECS cluster as an ECS container into a private subnet in the same VPC and private subnet with your Apache Kafka® infrastructure. The Lenses container does not have direct internet access, or a public IP address. Lenses outbound traffic must go out via a NAT gateway, and recipients of requests from the Lenses container will just see the request originating from the IP address of the NAT gateway.

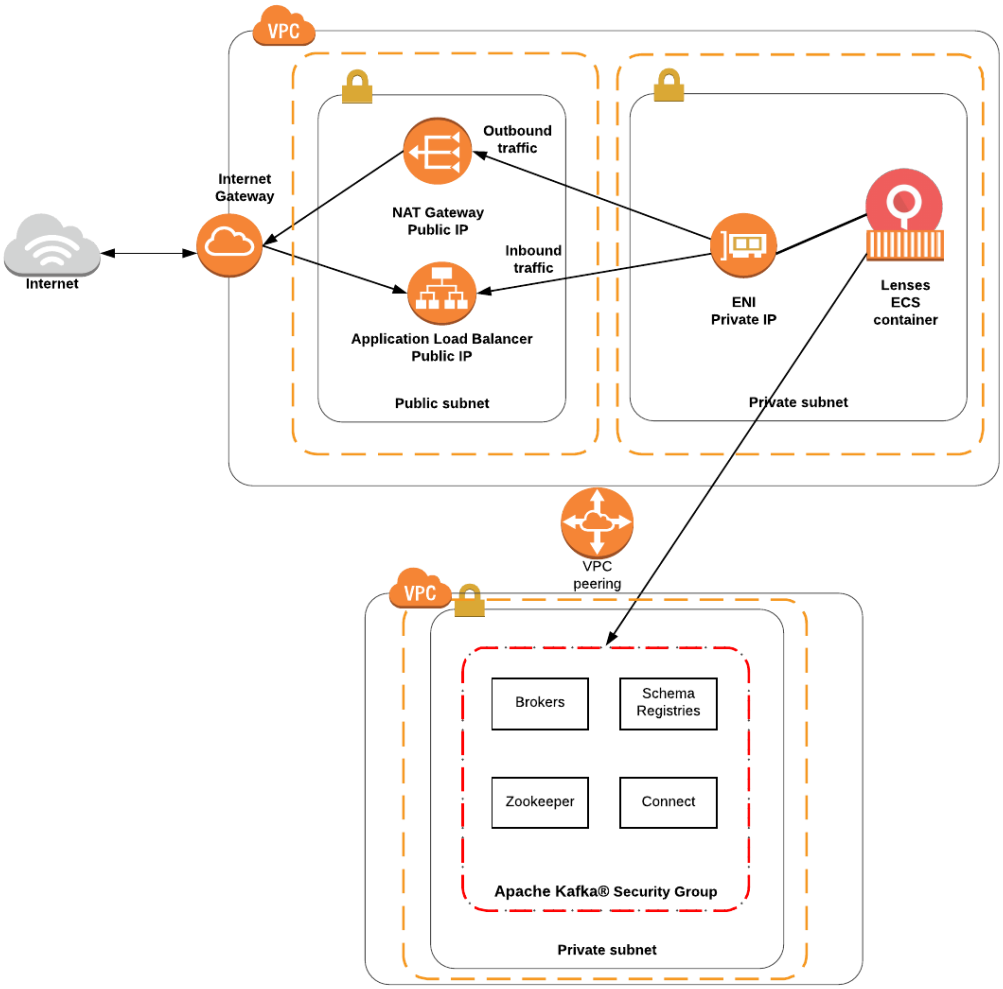
Lenses will be able to access your Apache Kafka® but you need to set a Security Group which allows the necessary Ingress traffic (allowed ports and source) to Brokers, Zookeepers, Schema Registries and Connect.



Lenses in ECS and Apache Kafka® in different VPC

The architecture is mostly the same with previous one but right now you have an Apache Kafka® in a completely different VPC and in a Private Subnet. Lenses will be able to have access to your Apache Kafka® by enabling a VPC peering between the two VPCs you have deployed Lenses and Apache Kafka® into.

Again you need to enable a Security Group to allow ingress traffic for the Brokers, Zookeepers, Schema Registries and Connect.



AutoDiscover Apache Kafka®

Lenses has the ability to autodiscover your own Apache Kafka® infrastructure in the cloud if you have give the proper permissions to the AWS IAM roles and also you have set the VPCs intercommunication.

Specifically if you have set proper tagging in your AWS resources as the following:

- Tag:Name: Broker, for your Apache Kafka® Broker
- Tag:Name: Zookeeper, for your Zookeeper
- TagName: Worker, for your Apache Kafka® Connect and Schema Registry

then Lenses discovers them in the Cloud and auto-generates the proper configuration. In order to support autodiscover of your Apache Kafka infrastructure, the AWS IAM role should include the `ec2:DescribeInstances` permission attached.

Logging

ECS

If you have deployed Lenses in an ECS cluster as an ECS container you can flow the logs of Lenses in CloudWatch. Your ECS Service Task Definition should have enabled the Logging Configuration and there should be assigned an AWS IAM role with the following permissions attached:

- `logs:CreateLogStream`
- `logs:PutLogEvents`

After this, your ECS container will send the logs to CloudWatch.

Monitoring

EC2

If you have deployed Lenses in an Amazon EC2 instance you can still use CloudWatch Monitoring but you need to have enabled the monitoring option.

ECS

If you have deployed Lenses in an Amazon ECS, you can monitor your Amazon ECS resources using Amazon CloudWatch, which collects and processes raw data from Amazon ECS into readable, near real-time metrics. These statistics are recorded for a period of two weeks, so that you can access historical information and gain a better perspective on how your Lenses cluster or service is performing. Amazon ECS metric data is automatically sent to CloudWatch in 1-minute periods.

Azure

Lenses in Azure

If you are using Azure you can easily deploy Lenses. Lenses can be deployed in Azure and pre-configured it with your current HDInsight Apache Kafka®.

In the cloud, security is really important and because Lenses access sensitive data it should not be exposed directly to the public Internet but there are a couple of secure ways to deploy Lenses in Cloud.

You can find Lenses in Azure Marketplace here.

Azure Resources

ARM template

Lenses can run in a VM instance using our own Azure Resource Manager templates which are available in the HDInsight and Azure marketplace. More details about the templates you can find here. During the deployment you need to fill in the form couple of information for your HDInsight Apache Kafka® or your own Apache Kafka®. Furthermore, you need to have your own license to use Lenses.

Deployment Scenarios

Azure offers great flexibility as such there's three main scenarios when deploying Lenses:

- Lenses as edge node of HDInsight Apache Kafka® (recommended)
- Lenses in VM and HDInsight Apache Kafka® in same VNET
- Lenses in VM and HDInsight Apache Kafka® with different VNETs and peering
- Lenses in VM and HDInsight Apache Kafka® with VPN gateway point-to-site configuration

Lenses as edge node of HDInsight Apache Kafka®

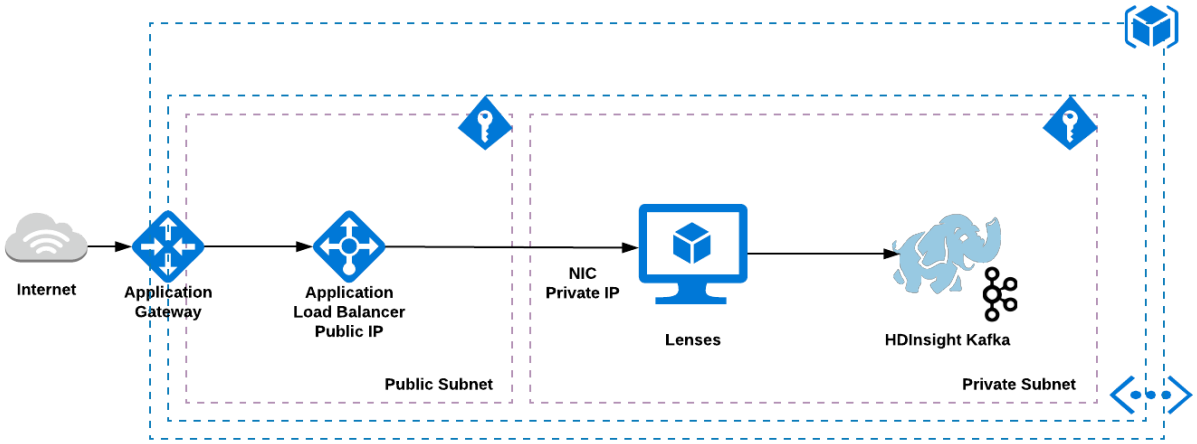
Running Lenses on an edge node created and managed directly by the Azure HDInsight cluster is the recommended deployment mode. In this case, HDInsight will create and configure the

edge node itself, and Lenses will be installed on this edge node as part of the HDInsight cluster. This kind of deployment will autodetect the Apache Kafka® Brokers and Zookeeper nodes, so the only required field you need to fill in is the Lenses license.

Azure HDInsight managed edge nodes are not visible from the Azure resource manager and thus can not leverage Azure persistent disks or other Azure native tools to perform automated backups. So if you use the Data Policies module for Lenses, you need to use your backup process to keep the storage data of Data Policies. You can check the directory you need to backup here.

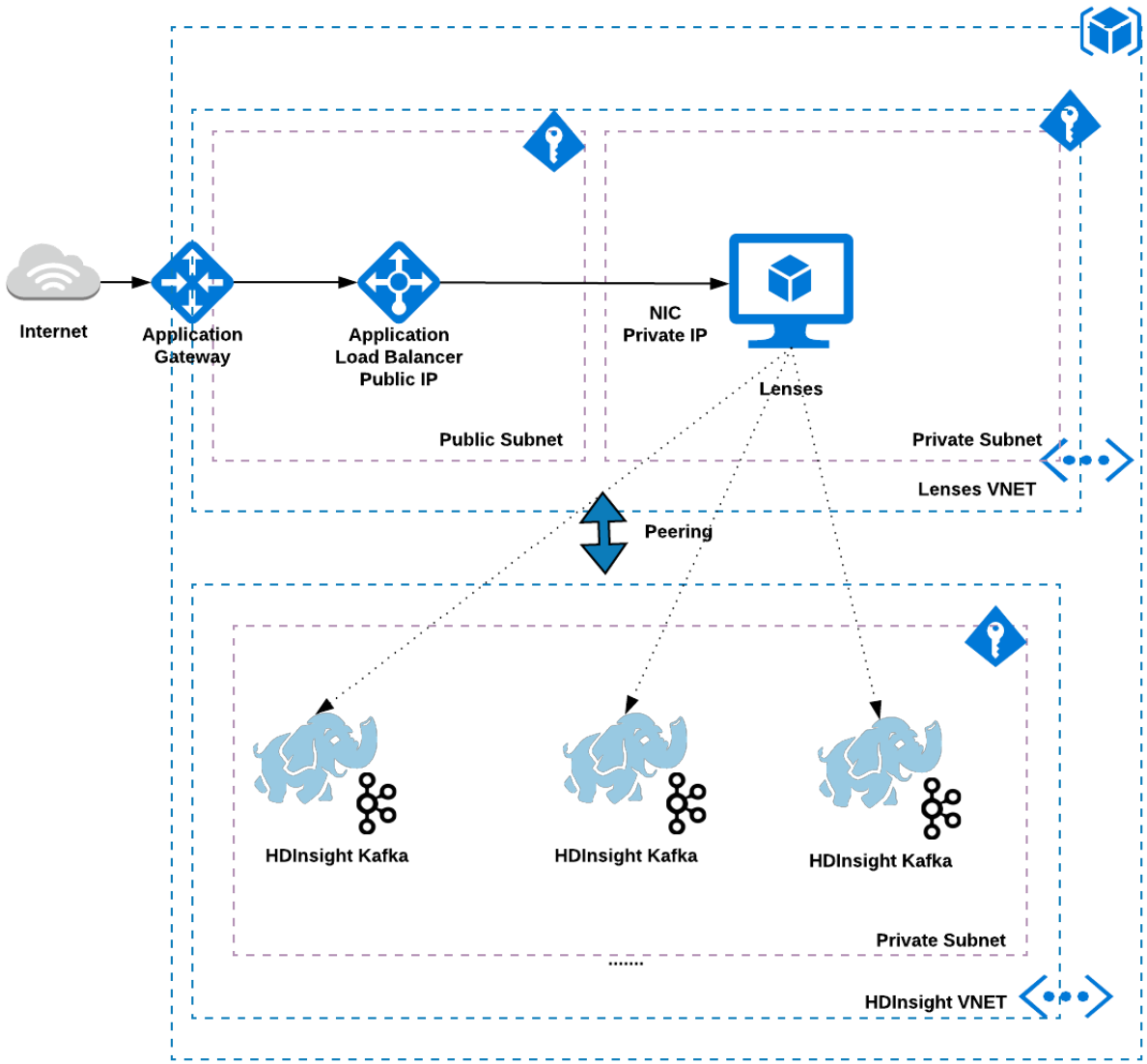
Lenses in VM and HDInsight Apache Kafka® in same VNET

This architecture deploys Lenses in a VM image into a private subnet in the same VNET and private subnet with your HDInsight Apache Kafka® infrastructure. The Lenses VM image does not have direct internet access, or a public IP address. Lenses outbound traffic must go out via a the the Application Gateway, and recipients of requests from the Lenses VM Image will just see the request originating from the IP address of the Application Gateway.



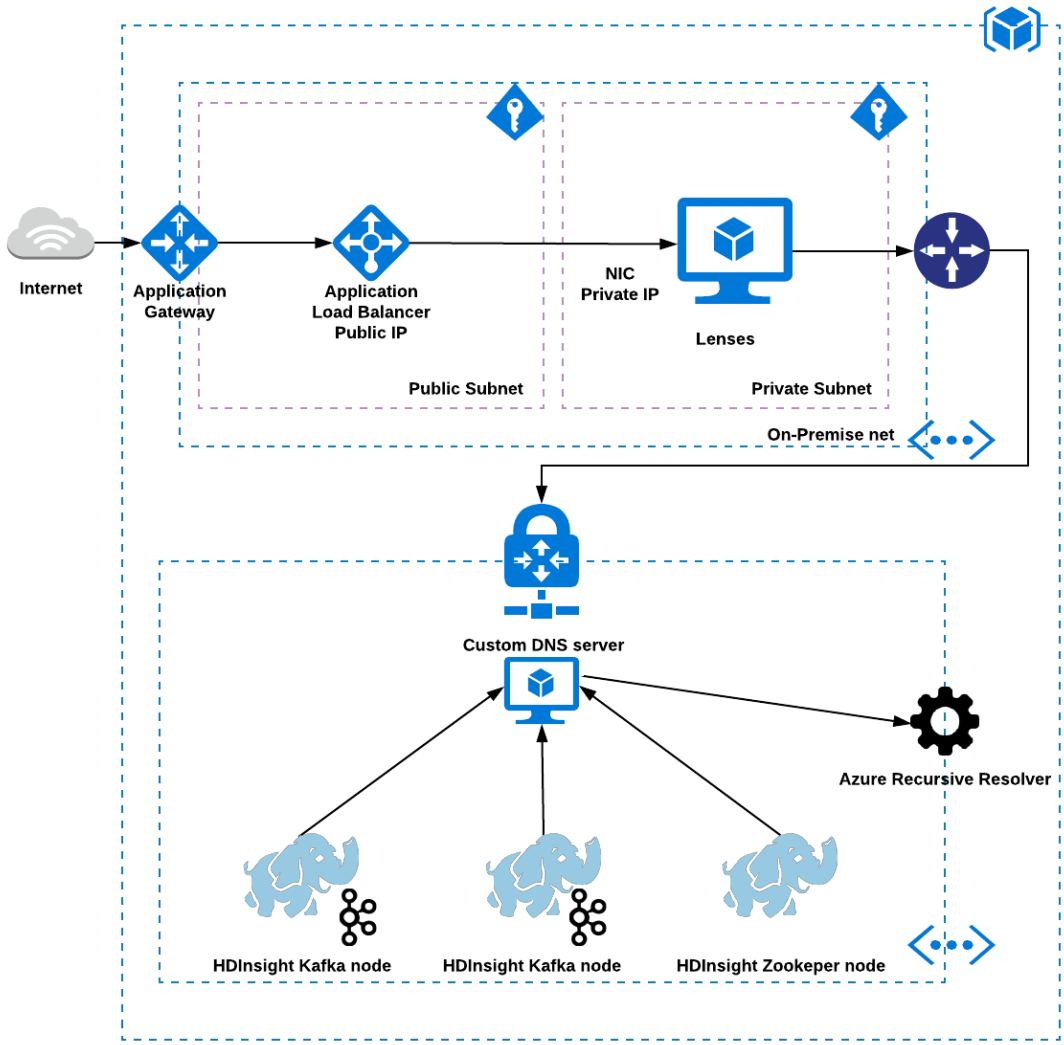
Lenses in VM and HDInsight Apache Kafka® in different VNETs and peering

This architecture deploys Lenses in a VM image into a private subnet in different VNETs with HDInsight Apache Kafka® infrastructure but with VNET Peering. With the VNET Peering Lenses can access HDInsight Apache Kafka® but in your cluster instead of domains you should advertise the IP of your brokers.



Lenses in VM and HDInsight Apache Kafka® with VPN site-to-site

This architecture deploys Lenses in an VM image into a private subnet in different VNETs with HDInsight Apache Kafka® infrastructure but with a VPN Gateway with site-to-site configuration.



GCP

Lenses in GCP

If you are using Google Cloud Platform (GCP) you can easily deploy Lenses. Lenses can be deployed in GCP and pre-configure it with your current Apache Kafka® infrastructure.

In the cloud, security is really important and because Lenses access sensitive data it should not be exposed directly to the public Internet but there are a couple of secure ways to deploy Lenses in Cloud.

GCP Resources

Deployment Manager

Lenses can run in a VM instance with a Container-Optimized OS using our own Deployment Manager templates for which you can find the details here. During the deployment you need to fill in the form couple of information for your Apache Kafka® cluster. Furthermore, you need to have your own license to use Lenses.

Deployment Scenarios

GCP offers great flexibility as such there's three main scenarios when deploying Lenses:

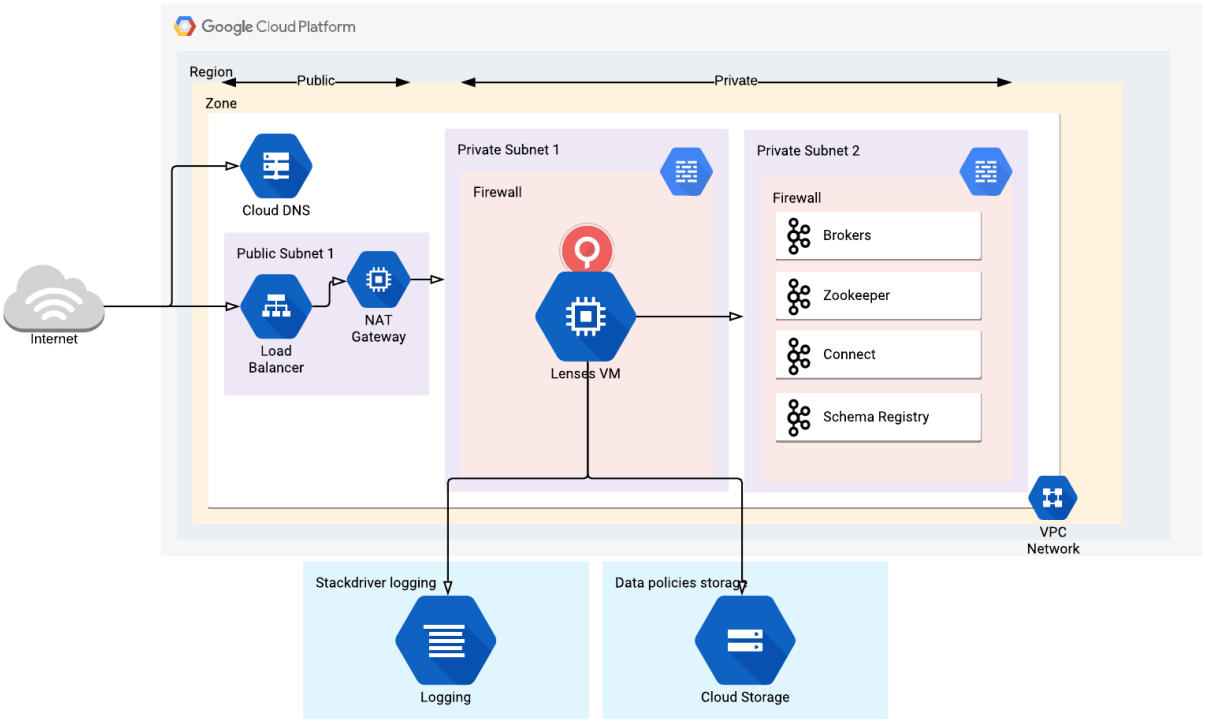
- Lenses in Container VM and Apache Kafka® in same VPC (recommended)
- Lenses in Container VM and Apache Kafka® with different VPCs and peering

Lenses in Container VM and Apache Kafka® in same VPC

This architecture deploys Lenses in a VM with a Container-Optimized OS into a private subnet in the same VPC and private subnet with your Apache Kafka® infrastructure in the same VPC bu in a different subnet. The Lenses VM does not have direct internet access, or a public IP address. Lenses outbound traffic must go out via a NAT gateway, and recipients of requests

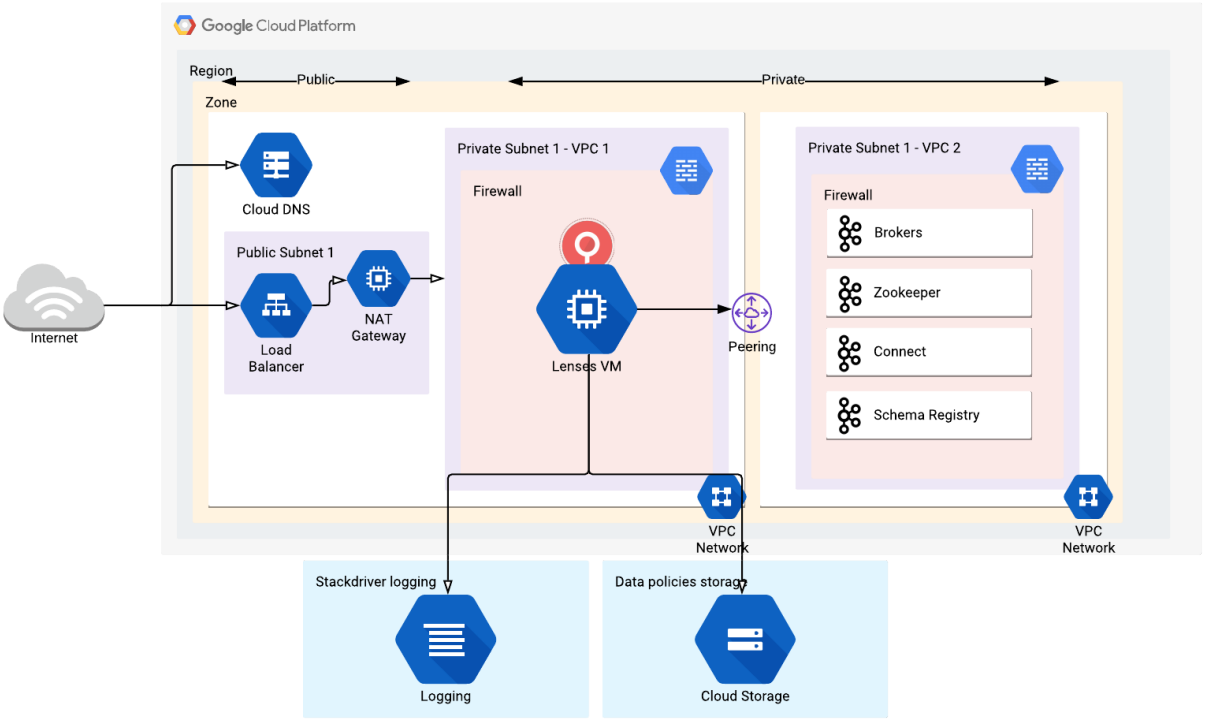
from the Lenses container will just see the request originating from the IP address of the NAT gateway.

Lenses will be able to access your Apache Kafka® but you need to set a Firewall Rule which allows the necessary Ingress traffic (allowed ports and source) to Brokers, Zookeepers, Schema Registries and Connect.



Lenses in Container VM and Apache Kafka® in different VPCs

This deployment scenario is exactly the same as the previous one but the key difference here is that Lenses is deployed in a different VPC than the Apache Kafka cluster which is deployed in its isolated VPC. The communication between these two VPCs happen with VPC peering.



Conclusion

This document provides a high level overview of the features provided by Lenses, empowering a data-driven enterprise. It describes the typical deployment patterns for Lenses in different scenarios.

Naturally different enterprises have varying requirements, we recommend working with our Landoop Professional Services for architectural reviews and guidance.

Give Lenses a spin at www.landoop.com
Contact us at info@landoop.com

About Us

Landoop creator of Lenses, is a London based company. Our aim is to help companies open up their business data to all relevant users seamlessly by giving them the power of data operations. DataOps is currently transforming data management. Building a data-driven culture mandates that all data personas work with data, enabling participation from the entire business. Organisations are trying hard to expose their data via platform teams. As a result, many end up with customised DIY solutions with in-house engineering teams spending most of their time building infrastructure and tooling. Data should be in the hands of its users, as simple as their email, to enable innovation and minimise time to value.

All rights reserved. Lenses is trademark and registered trademarks of Landoop Ltd. in the United Kingdom and other countries. All other brand names, product names, or trademarks belong to their respective owners.