Storage

- Single Store TSDB
  - Supported storage backends
- Chunk storage
  - File system
  - Object storage
    - Google Cloud Storage (GCS)
    - Amazon Simple Storage Storage (S3)
    - Azure Blob Storage
    - IBM Cloud Object Storage (COS)
    - Baidu Object Storage (BOS)
    - Alibaba Object Storage Service (OSS)
    - Other notable mentions
- Schema Config
- Upgrading Schemas
- Retention
- Examples
  - Single machine/local development (tsdb+filesystem)
  - GCP deployment (GCS Single Store)
  - AWS deployment (S3 Single Store)
  - Azure deployment (Azure Blob Storage Single Store)
    - Using account name and key
    - Using a service principal
  - IBM Deployment (COS Single Store)
  - On premise deployment (MinIO Single Store)

Unlike other logging systems, Grafana Loki is built around the idea of only indexing metadata about your logs: labels (just like Prometheus labels). Log data itself is then compressed and stored in chunks in object stores such as S3 or GCS, or even locally on the filesystem. A small index and highly compressed chunks simplifies the operation and significantly lowers the cost of Loki.

Loki 2.8 introduced TSDB as a new mode for the Single Store and is now the recommended way to persist data in Loki. This type only requires one store, the object store, for both the index and chunks. More detailed information about TSDB can be found under the manage section.

Single Store TSDB

Single Store refers to using object storage as the storage medium for both the Loki index as well as its data ("chunks"). There is one supported mode:

Starting in Loki 2.8, the TSDB index store improves query performance, reduces TCO and has the same feature parity as the deprecated "boltdb-shipper". TSDB is the recommended index store for Loki 2.8 and newer.

Supported storage backends

See Object Storage for supported backends.

Chunk storage

File system

The file system is the simplest backend for chunks, although it's also susceptible to data loss as it's unreplicated. This is common for single binary deployments though, as well as for those trying out loki or doing local development on the project. It is similar in concept to many Prometheus deployments where a single Prometheus is responsible for monitoring a fleet.

Object storage

Google Cloud Storage (GCS)

GCS is a hosted object store offered by Google. It is a good candidate for a managed object store, especially when you're already running on GCP, and is production safe.

Amazon Simple Storage Storage (S3)

S3 is AWS's hosted object store. It is a good candidate for a managed object store, especially when you're already running on AWS, and is production safe.

Azure Blob Storage

Blob Storage is Microsoft Azure's hosted object store. It is a good candidate for a managed object store, especially when you're already running on Azure, and is production safe. You can authenticate Blob Storage access by using a storage account name and key or by using a Service Principal.

IBM Cloud Object Storage (COS)

COS is IBM Cloud hosted object store. It is a good candidate for a managed object store, especially when you're already running on IBM Cloud, and is production safe.

Baidu Object Storage (BOS)

BOS is the Baidu Cloud hosted object storage.

Alibaba Object Storage Service (OSS)

OSS is the Alibaba Cloud hosted object storage.

Other notable mentions

You may use any substitutable services, such as those that implement the S3 API like MinIO.

Schema Config

Loki aims to be backwards compatible and over the course of its development has had many internal changes that facilitate better and more efficient storage/querying. Loki allows incrementally upgrading to these new storage *schemas* and can query across them transparently. This makes upgrading a breeze.

For a more detailed reference on schema configuration, including required values and the recommended settings for new installs, refer to Storage schema.

For instance, this is what it looks like when migrating from BoltDB with v11 schema to TSDB with v13 schema starting 2023-07-01:

```
schema_config:
  configs:
    - from: 2019-07-01
      store: boltdb-shipper
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: 24h
    - from: 2023-07-01
      store: tsdb
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h
```

For all data ingested before 2023-07-01, Loki used BoltDB with the v11 schema, and then switched after that point to the more effective TSDB with the v13 schema. This dramatically simplifies upgrading, ensuring it's simple to take advantage of new storage optimizations. These configs should be immutable for as long as you care about retention.

Upgrading Schemas

When a new schema is released and you want to gain the advantages it provides, you can! Loki can transparently query and merge data from across schema boundaries so there is no disruption of service and upgrading is easy.

First, you'll want to create a new period_config entry in your schema_config. The important thing to remember here is to set this at some point in the *future* and then roll out the config file changes to Loki. This allows the table manager to create the required table in advance of writes and ensures that existing data isn't queried as if it adheres to the new schema.

As an example, let's say it's 2023-07-14 and you want to start using the `v13` schema on the 20th:

```
schema_config:
  configs:
    - from: 2019-07-14
      store: tsdb
      object_store: filesystem
```

```
    schema: v11
    index:
      prefix: index_
      period: 24h
  - from: 2023-07-20
    store: tsdb
    object_store: filesystem
    schema: v13
    index:
      prefix: index_
      period: 24h
```

It's that easy; you just created a new entry starting on the 20th.

Retention

Loki manages retention through the Compactor when using TSDB. When retention is enabled, the Compactor identifies data that falls outside of the configured retention period, removes the corresponding index entries, and deletes the underlying chunk objects asynchronously.

For object storage backends (S3, GCS, Azure Blob) Loki no longer relies solely on external time to live (TTL) or bucket lifecycle rules; these may still be used as an additional safeguard, but Loki itself performs retention-driven deletion when configured.

When using the filesystem chunk store, Loki does not delete data based on disk usage or free-space conditions. Deletion is determined only by the retention settings, and disk-full scenarios must be handled operationally outside of Loki.

Loki also supports targeted deletion at the tenant or stream level.

For more information, see the retention configuration documentation.

Examples

{{< admonition type="note" >}} Loki uses the Thanos-based object storage clients by default, because the `use_thanos_objstore` setting defaults to `true`. With this default, Loki reads the configuration under `storage_config.object_store` and ignores the legacy client sections such as `storage_config.aws` or `storage_config.gcs`.

Each example below shows the Thanos configuration first. The legacy configuration follows it for reference, because the legacy clients are deprecated. To keep using a legacy client, you must set `use_thanos_objstore: false`.

To convert an existing configuration to the new format, refer to Migrate to Thanos storage clients.
{{< /admonition >}}

Single machine/local development (tsdb+filesystem)

The repo contains a working example, you may want to checkout a tag of the repo to make sure you get a compatible example.

GCP deployment (GCS Single Store)

Configuration using the Thanos-based object store client:

```
storage_config:
  use_thanos_objstore: true
  object_store:
    gcs:
      bucket_name: <BUCKET_NAME>
      service_account: |
        {
          "type": "service_account",
          ...
        }
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h # Can be increased for faster performance over longer query periods, uses
more disk space

schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: gcs
```

```
      schema: v13
      index:
        prefix: index_
        period: 24h
```

The same deployment using the deprecated GCS client:

```
storage_config:
  use_thanos_objstore: false # required to use the deprecated client
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
  gcs:
    bucket_name: <BUCKET_NAME>
    service_account: |
      {
        "type": "service_account",
        ...
      }

schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: gcs
      schema: v13
      index:
        prefix: index_
        period: 24h
```

`service_account` should contain JSON from either a GCP Console `client_credentials.json` file or a GCP service account key. If this value is blank, most services will fall back to GCP's Application Default Credentials (ADC) strategy. For more information about ADC, refer to How Application Default Credentials works.

The pre-defined `storage.objectUser` role (or a custom role modeled after it) contains sufficient permissions for Loki to operate.

{{< admonition type="note" >}} GCP recommends Workload Identity Federation instead of a service account key. {{< /admonition >}}

AWS deployment (S3 Single Store)

Configuration using the Thanos-based object store client:

```
storage_config:
  use_thanos_objstore: true
  object_store:
    s3:
      bucket_name: <BUCKET_NAME>
      # The endpoint is required. For AWS, use the regional S3 endpoint.
      endpoint: s3.<REGION>.amazonaws.com
      region: <REGION>
      # You can either declare the access key and secret in the config or
      # use environment variables AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY,
      # which will be picked up by the AWS SDK.
      access_key_id: <ACCESS_KEY_ID>
      secret_access_key: <SECRET_ACCESS_KEY>
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h         # Can be increased for faster performance over longer query
periods, uses more disk space

schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: s3
      schema: v13
      index:
        prefix: index_
        period: 24h
```

The Thanos-based client supports one bucket only. If you previously used `bucketnames` with several buckets, you must consolidate to a single bucket.

If you don't wish to hard-code S3 credentials, you can use an EC2 instance role instead. Leave `access_key_id` and `secret_access_key` unset. The client then looks for credentials in environment variables, in the AWS credentials file, and finally in the EC2 instance metadata:

```
storage_config:
  use_thanos_objstore: true
  object_store:
    s3:
      bucket_name: <BUCKET_NAME>
      endpoint: s3.<REGION>.amazonaws.com
      region: <REGION>
```

The same deployment using the deprecated S3 client:

```
storage_config:
  use_thanos_objstore: false # required to use the deprecated client
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
  aws:
    s3: s3://<ACCESS_KEY>:<URI_ENCODED_SECRET_ACCESS_KEY>@<REGION>
    bucketnames: <BUCKET_1>,<BUCKET_2>

schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: s3
      schema: v13
      index:
        prefix: index_
        period: 24h
```

To use an EC2 instance role with the deprecated client, change the `storage_config` section:

```
storage_config:
  use_thanos_objstore: false
  aws:
    s3: s3://<REGION>
    bucketnames: <BUCKET_1>,<BUCKET_2>
```

The role should have a policy with the following permissions attached.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LokiStorage",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::<ACCOUNT_ID>"
                ]
            },
            "Action": [
                "s3:ListBucket",
                "s3:PutObject",
                "s3:GetObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::<BUCKET_NAME>",
                "arn:aws:s3:::<BUCKET_NAME>/*"
            ]
        }
    ]
}
```

To setup an S3 bucket and an IAM role and policy:

This guide assumes a provisioned EKS cluster.

1. Checkout the Loki repository and navigate to production/terraform/modules/s3.
2. Initialize Terraform `terraform init`.
3. Export the AWS profile and region if not done so:

   ```
   export AWS_PROFILE=<AWS_PROFILE_NAME>
   export AWS_REGION=<EKS_CLUSTER_REGION>
   ```

4. Save the OIDC provider in an environment variable:

```
oidc_provider=$(aws eks describe-cluster --name <EKS_CLUSTER_NAME> --query
"cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\/\///")
```

See the IAM OIDC provider guide for a guide for creating a provider.

5. Apply the Terraform module `terraform -var region="$AWS_REGION" -var cluster_name=<EKS_CLUSTER_NAME> -var oidc_id="$oidc_provider"`
   Note, the bucket name defaults to `loki-data` but can be changed via the `bucket_name` variable.

Azure deployment (Azure Blob Storage Single Store)

Using account name and key

Configuration using the Thanos-based object store client:

```
schema_config:
  configs:
    - from: "2020-12-11"
      index:
        period: 24h
        prefix: index_
      object_store: azure
      schema: v13
      store: tsdb
storage_config:
  use_thanos_objstore: true
  object_store:
    azure:
      # Your Azure storage account name
      account_name: <ACCOUNT_NAME>
      # See https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-
introduction#containers
      container_name: <CONTAINER_NAME>
      # For the account key, see https://docs.microsoft.com/en-
us/azure/storage/common/storage-account-keys-manage?tabs=azure-portal
      # If you leave the account key unset, Loki uses an Azure managed identity instead.
      account_key: <ACCOUNT_KEY>
```

```
      # Set this to use a user assigned managed identity. If you leave it empty,
      # Loki uses the system assigned identity.
      user_assigned_id: <USER_ASSIGNED_IDENTITY_ID>
      # Configure this if you use a private Azure cloud, such as Azure Stack Hub.
      # Loki composes the storage URL as
https://account_name.endpoint_suffix/container_name/blob_name
      endpoint_suffix: <ENDPOINT_SUFFIX>
      # If `connection_string` is set, the `account_name` and `endpoint_suffix` values are not
used.
      # Use this instead of `account_key` to authenticate with a SAS token, or to use the
Azurite emulator.
      connection_string: <CONNECTION_STRING>
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
```

The same deployment using the deprecated Azure Blob Storage client:

```
schema_config:
  configs:
    - from: "2020-12-11"
      index:
        period: 24h
        prefix: index_
      object_store: azure
      schema: v13
      store: tsdb
storage_config:
  use_thanos_objstore: false # required to use the deprecated client
  azure:
    # Your Azure storage account name
    account_name: <ACCOUNT_NAME>
    # For the account-key, see docs: https://docs.microsoft.com/en-
us/azure/storage/common/storage-account-keys-manage?tabs=azure-portal
    account_key: <ACCOUNT_KEY>
    # See https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-
introduction#containers
    container_name: <CONTAINER_NAME>
    use_managed_identity: <TRUE|FALSE>
```

```
    # Providing a user assigned ID will override use_managed_identity
    user_assigned_id: <USER_ASSIGNED_IDENTITY_ID>
    request_timeout: 0
    # Configure this if you are using private azure cloud like azure stack hub and will use this endpoint suffix to compose container and blob storage URL. Ex: https://account_name.endpoint_suffix/container_name/blob_name
    endpoint_suffix: <ENDPOINT_SUFFIX>
    # If `connection_string` is set, the values of `account_name` and `endpoint_suffix` values will not be used. Use this method over `account_key` if you need to authenticate via a SAS token. Or if you use the Azurite emulator.
    connection_string: <CONNECTION_STRING>
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
  filesystem:
    directory: /loki/chunks
```

Using a service principal

The Thanos-based object store client has no service principal settings in `storage_config`. Instead, pass the credentials in the standard environment variables read by the Azure Identity Client Module for Go, which are `AZURE_TENANT_ID`, `AZURE_CLIENT_ID`, and `AZURE_CLIENT_SECRET`. Leave `account_key` unset so that Loki uses those credentials. You must still set `account_name`, because Loki uses it to build the storage URL.

```
schema_config:
  configs:
    - from: "2020-12-11"
      index:
        period: 24h
        prefix: index_
      object_store: azure
      schema: v13
      store: tsdb
storage_config:
  use_thanos_objstore: true
  object_store:
    azure:
      account_name: <ACCOUNT_NAME>
```

```
      container_name: <CONTAINER_NAME>
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
```

The same deployment using the deprecated Azure Blob Storage client:

```
schema_config:
  configs:
    - from: "2020-12-11"
      index:
        period: 24h
        prefix: index_
      object_store: azure
      schema: v13
      store: tsdb
storage_config:
  use_thanos_objstore: false # required to use the deprecated client
  azure:
    use_service_principal: true
    # Azure tenant ID used to authenticate through Azure OAuth
    tenant_id : <TENANT_ID>
    # Azure Service Principal ID
    client_id: <CLIENT_ID>
    # Azure Service Principal secret key
    client_secret: <CLIENT_SECRET>
    # See https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-
introduction#containers
    container_name: <CONTAINER_NAME>
    request_timeout: 0
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
  filesystem:
    directory: /loki/chunks
```

IBM Deployment (COS Single Store)

{{< admonition type="note" >}} The Thanos-based object store client does not support IBM COS. Because `use_thanos_objstore` defaults to `true`, you must set it to `false` to use COS. If you leave the default in place, Loki fails to start with the error `unrecognized object_store type cos`.

For the list of backends that the Thanos-based client supports, refer to the Thanos storage configuration reference. {{< /admonition >}}

```
schema_config:
  configs:
    - from: 2020-10-01
      index:
        period: 24h
        prefix: loki_index_
      object_store: cos
      schema: v13
      store: tsdb

storage_config:
  use_thanos_objstore: false # COS is not supported by the Thanos-based client
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
  cos:
    bucketnames: <BUCKET_1>, <BUCKET_2>
    endpoint: <ENDPOINT>
    api_key: <API_KEY_TO_AUTHENTICATE_WITH_COS>
    region: <REGION>
    service_instance_id: <COS_SERVICE_INSTANCE_ID>
    auth_endpoint: <IAM_ENDPOINT_FOR_AUTHENTICATION>
```

On premise deployment (MinIO Single Store)

You configure MinIO by using the S3 settings, because MinIO implements the S3 API.

Configuration using the Thanos-based object store client:

```
storage_config:
  use_thanos_objstore: true
  object_store:
    s3:
      bucket_name: <BUCKET_NAME>
      # Use a fully qualified domain name (fqdn), like localhost, without a scheme.
      endpoint: <FQDN>:<PORT>
      access_key_id: <USERNAME>
      secret_access_key: <SECRET>
      insecure: true            # set to false if MinIO is served over https
      bucket_lookup_type: path  # MinIO's equivalent of s3forcepathstyle
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h         # Can be increased for faster performance over longer query
periods, uses more disk space

schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: s3
      schema: v13
      index:
        prefix: index_
        period: 24h
```

The same deployment using the deprecated S3 client:

```
storage_config:
  use_thanos_objstore: false # required to use the deprecated client
  aws:
    # Note: use a fully qualified domain name (fqdn), like localhost.
    # full example: http://loki:supersecret@localhost.:9000
    s3: http(s)://<USERNAME>:<SECRET>@<FQDN>:<PORT>
    s3forcepathstyle: true
  tsdb_shipper:
    active_index_directory: /loki/index
    cache_location: /loki/index_cache
    cache_ttl: 24h
```

```
schema_config:
  configs:
    - from: 2020-07-01
      store: tsdb
      object_store: s3
      schema: v13
      index:
        prefix: index_
        period: 24h
```

□□□ #2
□ □□□ □□□ 2026-10-31 13:44:39 UTC
□ □□□ □□□ 2026-10-31 13:44:40 UTC