**4/29/2021**

[**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_get\_started\_tutorial\_intro.htm**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_intro.htm)

[**https://github.com/tableau**](https://github.com/tableau)

**Server**

1. [**https://help.tableau.com/current/server/en-us/saml\_requ.htm**](https://help.tableau.com/current/server/en-us/saml_requ.htm)
2. [**https://help.tableau.com/current/pro/desktop/en-us/useracct.htm**](https://help.tableau.com/current/pro/desktop/en-us/useracct.htm)
3. **Concept**

**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_concepts.htm**

1. **https://10ax.online.tableau.com/#/site/apiautodev705901/home**

This is the introduction to a 3-part tutorial on the basics of using the Tableau Server REST API to **manage your resources**. With this content, you can learn to use the REST API to:

* Sign in to a server.
* Create a project and publish a **workbook** to it.
* Create a user group and give it permissions to your workbook.
* **Introduction and Set Up** (this page)
  + Join the Tableau Developer Program to acitvate your free Tableau sandbox and more (or use your own Tableau server).
  + Set up the tools used for the tutorial.
  + Download the tutorial workbook file (or use your own).
* [**Part 1: Tools, REST Basics, and Sign In**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_part_1.htm) (~20 minutes)
  + Read about REST basics.
  + Make a REST request to sign in to your server.
  + Learn to use the server’s response to authenticate other requests.
* [**Part 2: Create a Project and Publish a Workbook to It**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_part_2.htm) (~20 minutes)
  + Create a Tableau project.
  + List the projects on your site.
  + Publish a workbook.
* [**Part 3: Create a User Group and Give it Permissions**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_part_3.htm) (~20 minutes)
  + Manually add user to your site.
  + Create a group.
  + Add users to your group.
  + Give users permissions to your workbook.

[**https://www.tableau.com/developer/confirmation**](https://www.tableau.com/developer/confirmation)

* Testing and Troubleshooting REST API Calls

<https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_testing.htm>

### **Examine log files**

Requests that you make to Tableau Server using the REST API result in log file entries on the server. You can examine the following log files to help troubleshoot REST API issues:

* *log-file-location*\httpd\access.*date-time-stamp*.log (for example, access.2015\_05\_11\_00\_00\_00.log).
* *log-file-location*\logs\vizportal\vizportal-*process-number*.log (for example vizportal-0.log)
* **Why would I want to use the REST API to manage my Tableau Server and resources?**
* The Tableau Server REST API enables you to create a script or program that performs the same actions you can take through the server UI.
* That means you **can automate repetitive tasks**, create automated workflows that behave differently based on the condition of your Tableau resources, integrate Tableau management tasks into your existing workflows, and more.

[**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_get\_started\_tutorial\_part\_1.htm**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_part_1.htm)

## **Authentication of REST calls**

Here's one more important piece of information about REST that you will use in all REST requests you make once you have signed in to a server.

Tableau Server responds to a sign in call with **a credentials token**. The token is used by the server to validate that a request comes from a signed-in user. When a token expires, **typically after 2 hours**, you will need to make a new sign in request to get a fresh token.

Every REST call to the server after sign in needs to pass a valid token to succeed. To pass the token you place it in the request header as a key/value pair:

|  |  |
| --- | --- |
| key | Value (example) |
| X-Tableau-Auth | fDgkilutQqmJn0znEZr7Sg|0CHx3E9Sird1cOHv4yNVK86Y7opMmhKz |

Here is the model of a REST call to **create a new project** in a site, note that using the verb POST causes the project to be created:

POST https://*MY\_SERVER*/api/*api-version*/sites/*site-id*/projects

The *site-id* is the value from your sign in response.

## **Step 3: Publish a workbook**

To publish a workbook, use the POST verb.

### **URI**

Here is the model of a REST call to list publish a workbook to a site:

POST https://*MY\_SERVER*/api/*api-version*/sites/*site-id*/workbooks

[**https://help.tableau.com/current/server/en-us/sites\_addusers.htm**](https://help.tableau.com/current/server/en-us/sites_addusers.htm)

[**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_get\_started\_tutorial\_part\_3.htm**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_get_started_tutorial_part_3.htm)

Here is the model for the REST call to create a group:

POST /api/3.4/sites/*site-id*/groups

[Tableau Server Client (Python)](https://tableau.github.io/server-client-python/)

[**https://tableau.github.io/server-client-python/docs/**](https://tableau.github.io/server-client-python/docs/)

# Fundamentals of the Tableau Server REST API

# <https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_fundamentals.htm>

Tableau Server provides an application programming interface (API) that lets you **programmatically manage users, workbooks, data connections, and other resources on the server.** By using the API, you can create users or import them from Active Directory, publish workbooks, create, view, and delete data sources, and perform other actions on the server.

Using the API, you can perform many of the tasks that you can do using [tabcmd](https://help.tableau.com/current/server/en-us/help.htm" \l "tabcmd.htm). However, the **REST API methods provide more granular control over your interaction with the server**. You can think of them as a set of programmatic blocks that you **can use to put together complex operations that chain the output of one operation to the input of the next one**, and that might involve conditions and other scenarios that are best addressed in programming logic.

In REST, resources are identified in a consistent way using a URI (uniform resource identifier). Actions are expressed using standard verbs like GET and POST. The client passes all necessary information to the server for each action—that is, the server does not have to maintain any state about the client

The URI specifies the site, project, workbook, user, or other resource that you are creating, viewing, or deleting

https://*your-server*/api/3.11/sites/*site-id*/groups/*group-id*/users

### **Specifying Resources on Tableau Online**

For **scalability and reliability**, the Tableau Online infrastructure extends over **multiple server instances**. In this environment, each site is assigned to a distinct instance (or pod). When you sign in using online.tableau.com, the site redirects you to the instance where your site resides. You will see the redirect reflected in the URL shown in the browser’s address bar. For example, instead of online.tableau.com, it might show the following address:

**https://10ay.online.tableau.com**

When you make REST API calls to Tableau Online, rather than using online.tableau.com, you need to use the URL for the instance on which your site exists.

The following URI specifies an individual project on a site on the 10ay instance:

https://10ay.online.tableau.com/api/3.11/sites/*site-id*/projects/*project-id*

## **Passing Additional Information in the Query String**

For some calls, you pass parameters using the query string. These parameters indicate processing instructions for the request, as distinct from data. One example is the **overwrite** query string parameter that you can include when you publish a workbook, to indicate that Tableau should overwrite any existing workbook. Another example is pagination instructions. The following example shows the URI for a GET request to list all the sites on the server. The parameters in the query string specify that the request should return elements 51 through 100.

https://*my-server*/api/3.11/sites?pageSize=50&pageNumber=2

Some methods also let you pass parameters in the query string that let you filter the results to return. For example, when you call [Get Users on Site](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#get_users_on_site), you can add the **filter** parameter and a filter expression like the following to the URI in order to return only users whose site role is Viewer:

https://MY-SERVER/api/3.11/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users?filter=siteRole:eq:Viewer

For some methods, you can also add the **sort** parameter and a sort expression to the query string, as in this example:

https://MY-SERVER/api/3.11/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users?sort=name:desc

For some methods, you can add the **fields**parameter to qualify the results by the fields and resources you are interested in. For example, when you call [List Metrics for Site](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#Query), you can add the **fields**parameter and a field expression to return just the id and name of all the workbooks:

https://localhost/api/2.5/sites/1a10f5b9-029b-43e4-a620-773d1690338c/workbooks?fields=id,name

For more information, see [Filtering and Sorting in the Tableau REST API](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_filtering_and_sorting.htm) and [Using Fields in the REST API](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_fields.htm).

**Sign In**

POST /api/2.2/auth/signin HTTP/1.1

HOST: my-server

Content-Type:application/json

Accept:application/json

{

"**credentials**": {

"name": "administrator",

"password": "passw0rd",

"site": {

"contentUrl": ""

}

}

}

**Get a list of resources**

The request must include an **X-Tableau-Auth header** with the authentication **token** that was returned in the [Sign In](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#sign_in) call. There is no request body.

The following examples shows a request that gets a list of users.

GET /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users/users HTTP/1.1

HOST: my-server

**X-Tableau-Auth:** 12ab34cd56ef78ab90cd12ef34ab56cd

Create a new resource

To create a new resource, such as a new user, you send a POST request. The request must include:

* An X-Tableau-Auth header with the authentication token that was returned in the [Sign In](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#sign_in) call
* A Content-Type header set to text/xml or application/xml if you are sending an XML block, or set to application/json for a JSON request block.
* A request body with an XML or a JSON block that includes the information for the new resource. The XML and JSON block is defined by an [XML schema](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_schema.htm). The information in the request block depends on what type of resource you're creating.

The following example shows a request that creates a new user in an XML request. The response returns the ID of the resource you created.

POST /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users HTTP/1.1

HOST: my-server

X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd

Content-Type: text/xml

<tsRequest>

<user name="NewUser" siteRole="Publisher" />

</tsRequest>

This example shows the same request in JSON.

POST /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users HTTP/1.1

HOST: my-server

**X-Tableau-Auth**: 12ab34cd56ef78ab90cd12ef34ab56cd

Content-Type: application/json

{

"user": {

"name": "NewUser1",

"siteRole": "Publisher"

}

}

Update a resource

To update an existing resource, you send a PUT request and include the ID of the resource that was returned in POST request. The request must include:

* An X-Tableau-Auth header with the authentication token that was returned in the [Sign In](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#sign_in) call.
* A Content-Type header set to text/xml.
* A request body with an XML or JSON block that includes the changes for the existing resource. The XML or JSON block is defined by an [XML schema](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_schema.htm). The information in the request block depends on the type of resource that you're updating.

The following example shows a request that updates an existing user. The URI includes the ID of the user that was returned from the POST request.

PUT /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users/9f9e9d9c-8b8a-8f8e-7d7c-7b7a6f6d6e6d HTTP/1.1

HOST: my-server

X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd

Content-Type: text/xml

<tsRequest>

<user fullName="NewUser2" siteRole="ViewerWithPublish" />

</tsRequest>

This example shows the same request in JSON.

PUT /api/2.2/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users/9f9e9d9c-8b8a-8f8e-7d7c-7b7a6f6d6e6d HTTP/1.1

HOST: my-server

X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd

Content-Type: application/json

{

"user": {

"fullName": "NewUser2",

"siteRole": "ViewerWithPublish"

}

}

**Delete a resource**

To remove resource, you send a DELETE request. The request must include an X-Tableau-Auth header with the authentication token that was returned in the [Sign In](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#sign_in) call. There is no request body.

The following example shows a request that deletes an existing user.

DELETE /api/2.2/sites/d0356794-bb9d-4c5c-b43d-ec384a2baf5a/users/2798bf2f-964d-4cf6-994a-0744c4555f84 HTTP/1.1

HOST: my-server

X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd

**Publish a resource**

To publish a resource such as a data source or a workbook, you have two options. You can publish the resource using a single call. In that case, you create a POST request with the content type of multipart/mixed, and you include the resource in the body of the request.

Another option is to publish the resource in pieces. For that scenario, you initiate a file upload as a POST request, and then send a series of POST requests that each append another piece of the resource. When the resource has been completely uploaded, you make a final POST request to finish the upload.

# REST API and Resource Versions

*Version: 2021.1*

## **Two Version Types**

**Classic:** **The more than 150 REST endpoints** that let you programmatically take many of the actions available in the user interface of Tableau Online, on-premise Tableau Server, Tableau Prep and more. This powerful set of APIs started life in Tableau Server 2.5 and has been evolving organically as our server products grew and matured. The existing REST APIs are in heavy usage throughout the Tableau community and will continue to be maintained and grow.

Endpoints of the classic Tableau REST API have request URIs that begin with **{server}/api/{api-version-number}**. The API version defines a set of resource versions that ship with a given release of Tableau Server (such as 2019.4). Versions of the REST API are identified using **major and minor version** numbers in the format **<major>.<minor>**, as **in 3.6**. For more information, see [REST API Versioning](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_versions.htm#rest_api_versioning).

**New RESTful endpoints:** Alongside the classic APIs, starting with Tableau Server 2020.2 we will be releasing a growing body of a new kind of endpoint. Because the new APIs are based a modern standard and designed for the mature Tableau environment, our teams will be able to produce more consistent endpoints at a faster pace. The new endpoints will call the same code that clicking the corresponding UI element calls, not always the case for classic APIs. URIs for new generation of endpoints begin with **{server}/api/-** where the version number is replaced with a hyphen. These resources are versioned independently of the REST API version number. For more information, see [Per Resource Versioning](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_versions.htm#per_resource_versioning).

REST API Versioning

When you make a request using the REST API, you include the API version number as part of the request, as in the following example:

https://MY-SERVER/api/3.11/auth/signin

When a new version of the REST API becomes available, if you try to call a method that was introduced in the new version but the URI includes a previous version, **the method call will fail**. In the [API reference documentation](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm), each method includes a **Version** section that tells you the version of the REST API in which that method was introduced

REST API Update releases

Small changes to the REST API might be made during update releases, which are identified using the format *<major>.<minor>.<update-release>*. For example, version 2.0.1 is an update release for version 2.0 of the REST API. Update releases might include bug fixes, and might include a new element or attribute in the XML body for a method.

When you specify the API version in the URI of a request, do not include the update release value. For example, when you specify the version number in the URI for update release 2.0.1, use version 2.0 in request URIs

# Signing In and Signing Out (Authentication)

*Version: 2021.1*

The Tableau Server REST API requires that you send a credentials token with each request. The credentials token lets the server verify you as a valid, signed in user. **To get a token**, you call [**Sign In**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#Sign) and pass credentials of a valid user, either a Personal Access Token (PAT) or a user name and password, along with the content URL (subpath) of the site you are signing in to.

SAML **single sign on (SSO) authentication** does not validate REST API requests. Even if you are manually signed in to your server through SSO, **REST API request authentication requires that you first make a REST sign in request, and then use the credentials token** from its response in the header of subsequent requests

Sign In URI

The following example shows the URI for a **Sign In** using a POST request:

POST http://my-server/api/3.11/auth/signin

**Server path parameter**

The my-server value in the sign in URI is the base URL for your Tableau Server. For Tableau Online, the server address in the URI must contain the pod name, such as 10az, 10ay, or us-east-1. For example, the URI to sign in to a site in the 10ay pod would be:

https://10ay.online.tableau.com/api/3.11/auth/signin

**Using Personal Token**

[**https://help.tableau.com/v2019.4/server/en-us/security\_personal\_access\_tokens.htm**](https://help.tableau.com/v2019.4/server/en-us/security_personal_access_tokens.htm)

## **Make a Sign In Request with a Personal Access Token**

Personal Access Token (PAT) is used to acquire an **access token**, without using username/passwd.

A Personal Access Token (PAT) validates that a user is allowed to sign in to a site. A sign in using a PAT returns the same info as a username and password sign in (a credentials token, site LUID, and user LUID), but without the security risk of exposing hard-coded usernames and passwords, or an interactive login experience.

PATs are **long-lived**, and can be **revoked** without disabling the Tableau user they are attached to. **A user can have multiple PATs**, which allows for granular monitoring and revocation of access rights. For instance, an admin might group their scripts into functional areas (like permissions, subscriptions, and data source refreshes) and use a different PAT for each area. For more information, see [Personal Access Token(Link opens in a new window)](https://help.tableau.com/v2019.4/server/en-us/security_personal_access_tokens.htm).

Personal Access Tokens will expire if they are not utilized **for 15 consecutive days**. If they are regularly used more frequently than every 15 days, an access token will expire after 1 year, and need to be replaced with a newly created one.

The tokens allow users to run automation with Tableau REST APIs without requiring hard-coded credentials or interactive login.

When you [create a Personal Access Token(Link opens in a new window)](https://help.tableau.com/v2019.4/pro/desktop/en-us/useracct.htm#create-and-revoke-personal-access-tokens), Tableau displays a dialog that shows the **token name** and **token secret**.

The sign-in request body using JSON looks like the following. Its header should contain Content : application/json.

{

"credentials": {

"personalAccessTokenName": "MY\_TOKEN\_NAME",

"personalAccessTokenSecret": "qlE1g9MMh9vbrjjg==:rZTHhPpP2tUW1kfn4tjg8",

"site": {

"contentUrl": "MarketingTeam"

}

}

}

Make a Sign In Request with Username and Password

{

"credentials": {

"name": "admin",

"password": "p@ssword",

"site": {

"contentUrl": "MarketingTeam"

}

}

}

## **Response for a Successful Sign In Operation**

If the **Sign In** call is successful, the body of the response contains **an authentication token**, the site ID of the site you're signed in to, and the **user ID** of the user you're signed in as

{

"credentials": {

"site": {

"id": "9a8b7c6d5-e4f3-a2b1-c0d9-e8f7a6b5c4d",

"contentUrl": ""

},

"user": {

"id": "9f9e9d9c-8b8a-8f8e-7d7c-7b7a6f6d6e6d"

},

"token": "12ab34cd56ef78ab90cd12ef34ab56cd"

}

}

When you get the response, you parse the token out of the response and store it in your application. By default, the token is good for 240 minutes.

(You can specify a different timeout value for the token by calling the [tsm configuration set](https://help.tableau.com/current/server/en-us/cli_configuration-set_tsm.htm) command to change the **wgserver.session.idle\_limit** setting.) If your application needs to be able to make additional calls after the token has expired, you can call **Sign In** again and get a new authentication token.

You include the authentication token as the value of the **X-Tableau-Auth** header for all other REST API calls. For example:

X-Tableau-Auth: 12ab34cd56ef78ab90cd12ef34ab56cd

When you are finished with a session, you call **Sign Out**. This invalidates the token, which makes sure that no one else can use the authentication token to make calls to the REST API.

# This example shows how to use the Tableau Server REST API  
# to sign in to a server, get back an authentication token and  
# site ID, and then sign out.  
# The example runs in Python 2.7 and Python 3.3 code  
  
import requests, json  
  
  
# NOTE! Substitute your own values for the following variables  
use\_pat\_flag = True # True = use personal access token for sign in, false = use username and password for sign in.  
  
server\_name = "YOUR\_SERVER" # Name or IP address of your installation of Tableau Server  
version = "x.x" # API version of your server  
site\_url\_id = "SITE\_SUBPATH" # Site (subpath) to sign in to. An empty string is used to specify the default site.  
  
# For username and password sign in  
user\_name = "USERNAME" # User name to sign in as (e.g. admin)  
password = "{PASSWORD}"  
  
# For Personal Access Token sign in  
personal\_access\_token\_name = "TOKEN\_NAME" # Name of the personal access token.  
personal\_access\_token\_secret = "TOKEN\_VALUE" # Value of the token.  
  
signin\_url = "**https**://{server}/api/{version}/auth/signin".format(server=server\_name, version=version)  
  
if use\_pat\_flag:  
 # The following code constructs the body for the request.  
 # The resulting element will look similar to the following example:  
 #  
 # {  
 # "credentials": {  
 # "personalAccessTokenName": "TOKEN\_NAME",  
 # "personalAccessTokenSecret": "TOKEN\_VALUE",  
 # "site": {  
 # "contentUrl": ""  
 # }  
 # }  
 # }  
 #  
  
 payload = { "credentials": { "personalAccessTokenName": personal\_access\_token\_name, "personalAccessTokenSecret": personal\_access\_token\_secret, "site": {"contentUrl": site\_url\_id }}}  
  
 headers = {  
 'accept': 'application/json',  
 'content-type': 'application/json'  
 }  
  
else:  
 # The following code constructs the body for the request. The resulting element will# look similar to the following example:  
 #  
 #  
 # {  
 # "credentials": {  
 # "name": "USERNAME",  
 # "password": "PASSWORD",  
 # "site": {  
 # "contentUrl": ""  
 # }  
 # }  
 # }  
 #  
  
 payload = { "credentials": { "name": user\_name, "password": password, "site": {"contentUrl": site\_url\_id }}}  
  
 headers = {  
 'accept': 'application/json',  
 'content-type': 'application/json'  
 }  
  
# Send the request to the server  
req = requests.post(signin\_url, json=payload, headers=headers, verify=False)  
req.raise\_for\_status()  
  
# Get the response  
response = json.loads(req.content)  
  
# Parse the response JSON. The response body will look similar  
# to the following example:  
#  
# {  
# "credentials": {  
# "site": {  
# "id": "xxxxxxxxxx-xxxx-xxxx-xxxxxxxxxx",  
# "contentUrl": ""  
# },  
# "user": {  
# "id": "xxxxxxxxxx-xxxx-xxxx-xxxxxxxxxx"  
# },  
# "token": "CREDENTIALS\_TOKEN"  
# }  
# }  
#  
  
# Get the authentication token from the credentials element  
token = response["credentials"]["token"]  
  
# Get the site ID from the <site> element  
site\_id = response["credentials"]["site"]["id"]  
  
print('Sign in successful!')  
print('\tToken: {token}'.format(token=token))  
print('\tSite ID: {site\_id}'.format(site\_id=site\_id))  
  
# Set the authentication header using the token returned by the Sign In method.  
headers['X-tableau-auth']=token  
  
  
  
# ... Make other calls here ...  
  
  
# Sign out  
signout\_url = "https://{server}/api/{version}/auth/signout".format(server=server\_name, version=version)  
  
req = requests.post(signout\_url, data=b'', headers=headers, **verify=False**)  
req.raise\_for\_status()  
print('Sign out successful!')

Revoking users' Personal Aceess tokens

Users are able to revoke their own tokens on the **My Account Settings** page. As an administrator, you can also revoke personal access tokens.

1. [Sign in to the Tableau Server Admin Area](https://help.tableau.com/v2019.4/server/en-us/sign_in_server_admin.htm) as a site administrator or server administrator.
2. Locate the user whose token you want to revoke. For more information about navigating Server Admin pages and locating users, see [View, Manage, or Remove Users](https://help.tableau.com/v2019.4/server/en-us/users_view.htm).
3. Click the user's name to open their profile page.
4. On the user's profile page, click the **Settings** tab.
5. In the **Personal Access Tokens** section, identify the token that you want to revoke and then click **Revoke**.
6. On the verification pop-up, click **Delete**.

## **Tracking and monitoring usage**

All token-related actions are logged in the [Tableau Server Application Server](https://help.tableau.com/v2019.4/server/en-us/server_process_application-server.htm) (vizportal) service.

To locate token-related activities, filter log entries containing the string, RefreshTokenService.

Tokens are stored in this format :Token Guid: <TokenID(Guid)>, where the TokenID is a base64 encoded string. The token secret is not included in the logs. For example:

Token Guid: 49P+CxmARY6A2GHxyvHHAA== (e3d3fe0b-1980-458e-80d8-61f1caf1c700).

The following is an example snippet of two log entries. The first shows how a user is mapped to a token.The second shows a refresh event for the same token:

RefreshTokenService - Issued refresh token to the following user: jsmith. Token Guid: 49P+CxmARY6A2GHxyvHHAA== (e3d3fe0b-1980-458e-80d8-61f1caf1c700)

RefreshTokenService - Redeemed refresh token. Token Guid: 49P+CxmARY6A2GHxyvHHAA== (e3d3fe0b-1980-458e-80d8-61f1caf1c700)

To locate key operations, filter log entries containing the string, OAuthController.

# Permissions

# https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_concepts\_permissions.htm

*Version: 2021.1*

Users who call REST API methods must have permission to perform the task represented by each method that they call.

A user's set of permissions is determined by the user's identity—that is, by the credentials that the user passes when calling [**Sign In**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#sign_in).

Many methods, such as adding or removing users from the server, require the user to be a server administrator. Other methods can be called by any user who has been granted appropriate permissions to perform the task that is represented by the method. The documentation **for individual methods** indicates the **permissions** that are required in order to call that method.

Using the REST API to assign permissions

You can use methods in the REST API to assign or edit permissions for projects, workbooks, and data sources. For example, you might call [**Create Project**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#create_project) to create a new project and then call [**Add Project Permissions**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#add_project_permissions) to assign appropriate permissions to the new project. **When you add permissions to an entity like a project**, you specify the following information:

* The grantee, that is, the user or group that you are assigning the permission to. (A grantee is sometimes referred to as a *principal*.) Grantees are identified by ID (LUID), not name.
* The capability that you're assigning permissions to. You specify the capability using a reserved name that's defined in the REST API XML schema, such as **Read**, **Write**, **ExportData**, or **WebAuthoring**. (These reserved names correspond to names that are used in the server environment, as listed later in this topic.)
* The mode for the permission, which is either **Allow** or **Deny**.
* Individual entities (project, workbook, and data source) support different capabilities. For example, for a data source, you can assign permissions only for the **ChangePermissions**, **Connect**, **Delete**, **ExportXml**, **Read**, and **Write** capabilities. In contrast, for a workbook, you can assign permissions for the **AddComment**, **ChangeHierarchy**, **ChangePermissions**, **Delete**, **ExportData**, **ExportImage**, **ExportXml**, **Filter**, **Read**, **ShareView**, **ViewComments**, **ViewUnderlyingData**, **WebAuthoring**, and **Write** capabilities. The documentation for an individual API (such as [Add Workbook Permissions](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#add_workbook_permissions) or [Add Project Permissions](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#add_project_permissions)) specifies the capabilities that you can pass for that API.
* The following example shows what the body of an **Add Project Permissions** request might look like. This example assigns the **Allow** permission to the **Read** and **Write** capabilities of the project for the user whose ID is 9f9e9d9c-8b8a-8f8e-7d7c-7b7a6f6d6e6d.
* <tsRequest>  
   <permissions>  
   <granteeCapabilities>  
   <user id="9f9e9d9c-8b8a-8f8e-7d7c-7b7a6f6d6e6d" />  
   <capabilities>  
   <capability name="Read" mode="Allow" />  
   <capability name="Write" mode="Allow" />  
   </capabilities>  
   </granteeCapabilities>  
   </permissions>  
  </tsRequest>

# Using HTTPS (SSL/TLS) for API Calls

[**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_concepts\_SSL.htm**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_SSL.htm)

To make API calls with HTTPS, you must configure Tableau Server to use SSL. For details, see [SSL(Link opens in a new window)](https://help.tableau.com/current/server/en-us/help.htm#ssl_config.htm) in the Tableau Server documentation. (**If you are testing Tableau Server, you can install a self-signed certificate in order to support SSL**.) If you make an HTTPS call to Tableau Server but you have not configured the server to use SSL, the server refuses the request.

When working with Tableau Online, you must use the HTTPS protocol to make API calls.

SSL certificate requirements

Acquire an Apache SSL certificate from a trusted authority (for example, Verisign, Thawte, Comodo, GoDaddy). You can also use an internal certificate issued by your company. Wildcard certificates, which allow you to use SSL with many host names within the same domain, are also supported.

### **SSL with multiple gateways**

A highly available Tableau Server cluster can include multiple gateways, fronted by a load balancer. If you are configuring this type of cluster for SSL, you have the following choices:

* **Configure the load balancer for SSL**: Traffic is encrypted from the client web browsers to the load balancer. Traffic from the load balancer to the Tableau Server gateway processes is not encrypted. No SSL configuration in Tableau Server is required by you. It’s all handled by the load balancer.
* **Configure Tableau Server for SSL**: Traffic is encrypted from the client web browsers to the load balancer, and from the load balancer to the Tableau Server gateway processes. For more information, continue to the following section.

## **Filtering**

* By default, methods like **Get Users on Site** return information about all the users or workbooks on a Tableau Server site. If there are many entities, the method returns them in chunks (pages). If you want information about a specific user or workbook, one option is to loop through the information returned by the call and look for the user or workbook that you're interested in.
* As an alternative, starting in the REST API version 2.3 (Tableau Server 10.0), you can include a filter expression in the query string of the URI, using the following syntax:
* filter=*field*:*operator*:*value*
* The following example shows how to include a filter in a call to get users on a site:
* http://MY-SERVER/api/3.11/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users**?filter=siteRole:eq:Viewer**

### **Filter expressions**

The following table lists the field names and operators you can use in a filter expression. Notice that the user-related calls support slightly different fields than the workbook-related calls.

* Operators are delimited with colons (:).
* If any reserved characters following the question mark (?) in the URI are encoded, than all reserved characters must be encoded. For example, the colon character (:) would be encoded as %3A and the equals character (=) would be encoded as %3D.
* The operators are:
  + eq—equals
  + gt—greater than
  + gte—greater than or equal
  + has—contains the specified string
  + lt—less than
  + lte—less than or equal
  + in—any of [list] (for searching tags)
* Field names, operator names, and values are case sensitive.

## **Sorting**

To sort the results, include the sort parameter. The syntax is this:

sort=*field*:*direction*

You can sort on the same fields that you use to filter. The *direction* value is asc (ascending) or desc (descending), and is required.

You can specify multiple levels of sorting by including multiple sort expressions, as in this example:

sort=siteRole:asc,name:desc

This causes the results to be sorted first by site role in ascending order, and within site role, by name in descending order.

## **Combining parameters**

You can combine filter, sort, and paging operators in the same call, separating them with &. The following example shows how you can combine a filter parameter, a sort parameter, and a paging parameter.

http://MY-SERVER/api/3.11/sites/9a8b7c6d-5e4f-3a2b-1c0d-9e8f7a6b5c4d/users?filter=lastLogin:gte:2016-02-01T00:00:00Z&sort=siteRole:desc,name:desc&pageNumber=2

# Paginating Results

*Version: 2021.1*

REST API methods that might return a large number of individual elements support pagination. Some examples of these methods include [Query Datasources](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#query_datasources) and [Get Users in Site](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#get_users_in_site). When you call these methods, the response includes a **<pagination>** element that has **pageNumber**, **pageSize**, and **totalAvailable** attributes whose values you can use to implement paging.

The following example shows what the response body might look like for a **Get Users in Site** request where there are a total of 158 users in the site:

**I**f a method supports pagination, you can pass two parameters as part of the request: the page size (that is, the number of elements to return per call) and the page number. The default page size for an individual call is 100, but you can specify a page size that’s any positive integer. The page number is a 1-based (not zero-based) value. The default page number is 1, meaning that if you do not include a page number, Tableau Server returns the first page of elements.

You pass page number and page size parameters using the query string on the URI, as in this example:

http://*myserver*/api/3.11/sites/*site-id*/users?pageSize=20&pageNumber=2

try:  
 # Python 3  
 from urllib.request import urlopen, request  
except:  
 # Python 2  
 from urllib2 import urlopen, Request  
import xml.etree.ElementTree as ET # For parsing XML responses  
  
# ... Code here to sign in ...  
  
# Note: A Sign In request has already been made and the authentication token is  
# in the variable named "token"  
headers = {'X-tableau-auth': token}  
  
page\_size = 50  
page\_number = 1 # 1-based, not zero based  
total\_returned = 0  
done = False  
user\_name\_list = []  
  
while not(done):  
 request\_url = 'http://{server}/api/3.11/sites/{site\_id}/users'.format(  
 server=server, site\_id=site\_id)  
 paging\_parameters = '?pageSize={page\_size}&pageNumber={page\_number}'.format(  
 page\_size=page\_size, page\_number=page\_number)  
 full\_url = request\_url + paging\_parameters  
 print(full\_url)  
  
 # Make request  
 req = Request(full\_url, headers=headers)  
 req = urlopen(req)  
 server\_response = req.read().decode("utf8")  
  
 response\_xml = ET.fromstring(server\_response)  
  
 # Get total number of records from the <pagination> element  
 total\_available = response\_xml.find('.//t:pagination',  
 namespaces={'t': "http://tableau.com/api"}).attrib['totalAvailable']  
 # Note! Need to convert "total\_available" to integer  
 total\_available = int(total\_available)  
  
 page\_number += 1  
 total\_returned += page\_size  
  
 # Get user details from <users> element  
 user\_element = response\_xml.findall('.//t:user',  
 namespaces={'t': "http://tableau.com/api"})  
  
 # Display individual user names  
 for user in user\_element:  
 print (user.attrib['name'])  
  
 if(total\_returned >= total\_available):  
 done = True  
  
# ... Code here to sign out ...

Publishing Resources

*Version: 2021.1*

You can use the REST API to programmatically publish (**upload) workbooks and data sources to the server**. You can publish workbooks and data sources in two ways:

* Using a single call, in which the complete file to publish is sent in the body of the request.
* Using a **multi-part** upload, in which you send the file to the server in pieces. You should do this if the file you want to publish is very large, so that the upload process doesn't time out as you're publishing.

## **Publishing a resource by using a single call**

You can publish a file by using a single request, as long as the file is not too large. The maximum size of a file you can publish in one request is 64MB. However, if your file is that large, you should publish it in parts, as described under [Publishing Flow for Multi-Part Upload](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_publish.htm#multi_part) later in this topic.

To publish using a single call, call [Publish Workbook](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#publish_workbook) or [Publish Data Source](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#publish_workbook). These are POST requests that have the following format:

http://my-server/api/3.11/sites/*site-id*/workbooks

http://my-server/api/3.11/sites/*site-id*/datasources

In addition to the **X-Tableau-auth** header that you set for all requests, you must make sure that two other headers are set: **Content-Length** and **Content-Type**. The **Content-Type** header for the request must be set to multipart/mixed; boundary=*boundary-string*.

The boundary string is used to separate parts of the request body. The value of *boundary-string* can be any string up to 70 characters long that will not otherwise appear in the body of the request. One approach to managing boundary strings is to create a globally unique ID (GUID) and use that as the boundary string.

[**https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_concepts\_publish.htm**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_publish.htm)

Publishing flow for multi-part upload

If the file you want to publish is large, you should publish it in multiple parts (or, for workbooks, [publish asynchronously](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_publish.htm#async)) . If the file is larger than 64 MB, you must publish it in multiple parts. Publishing in parts prevents a request from timing out. To perform a multi-part upload, send a succession of requests to the server, consisting of these methods:

* [**Initiate File Upload**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#initiate_file_upload). Call this method to prepare the server to receive a file.
* [**Append to File Upload**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#append_to_file_upload). Call this method multiple times, passing a part of the file with each call.
* [**Publish Workbook**](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#publish_workbook) or [*Publish Data Source*](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_ref.htm#publish_data_source). Call one of these methods to commit the file upload and to assign a name and project to the resource you're publishing.

The first two parts of the upload process (initiating and appending) are the same whether you're publishing a workbook or a data source. It's only when you're ready to commit your upload that you call a method that is specific to a workbook or a data source.

Initiating the file upload

You begin a multi-part upload by calling the **Initiate File Upload** method. This is a POST request that has the following format:

http://my-server/api/3.11/sites/*site-id*/fileUploads

The response for this request is HTTP code **201 (Created)**. The response body includes a **<fileUpload>** element that includes an upload session ID:

<?xml version="1.0" encoding="UTF-8"?>

<tsResponse *version-and-namespace-settings* >

<fileUpload

**uploadSessionId**="13253:6744F321974F4E8B8EC1424A3D56E0EA-0:0" fileSize="0"/>

</tsResponse>

# Handling Errors in the REST API

*Version: 2021.1  
https://help.tableau.com/current/api/rest\_api/en-us/REST/rest\_api\_concepts\_errors.htm*

If a request results in an error, the server returns an HTTP status code that indicates the general nature of the error. These can include **400 (Bad Request)** if Tableau Server cannot parse or interpret the message in the request, **403 (Forbidden)** if the request was not authorized, **404 (Not Found)** if a resource could not be located, and **405 (Method Not Allowed)** if the wrong verb was used for an operation (for example, making a GET request instead of a POST request).

For error conditions, **the response body also includes an XML block that provides details about the error.** For example, if the HTTP response was 404, the response body provides details about what resource in particular was not found. Imagine that you send the following PUT request in order to update information for a user:

http://your-server/api/3.11/sites/9a8b7c6d5-e4f3-a2b1-c0d9-e8f7a6b5c4d/users/9f9e9d9c8-b8a8-f8e7-d7c7-b7a6f6d6e6d

An HTTP status code of 404 for the response tells you that the operation was not successful because a resource could not be found. In that case, the response body might look like the following example:

<?xml version="1.0" encoding="UTF-8"?>

<tsResponse *version-and-namespace-settings* >

<error code="404002">

<summary>Resource Not Found</summary>

<detail>User '9f9e9d9c8-b8a8-f8e7-d7c7-b7a6f6d6e6d' could not be found.</detail>

</error>

</tsResponse>

# Testing and Troubleshooting REST API Calls

# <https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_testing.htm>

## **Troubleshooting**

If REST API calls are not working the way you expect, you can troubleshoot them several ways.

### **Examine error codes**

If an error occurs during a request, the response includes both an HTTP status code (like 400, 403, or 404) and an <error> element in the response body. The <error> tag includes an error code that provides detailed information about the error. For information about errors, see [Handling Errors in the REST API](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_errors.htm). The documentation for individual methods in the API lists the error codes that a method might return.

### **Examine requests and responses**

It can be particularly helpful for troubleshooting to examine **the raw requests** that go between your client and the server. By examining **the contents of each request** (the **HTTP verb** and **URI**, **the headers**, and the request and response **bodies**) and **comparing those to the documentation**, you can often determine why a REST method is not working as it should.

For Windows, you can use [**Fiddler**(Link opens in a new window)](http://www.telerik.com/fiddler), which is a web proxy tool that lets you examine and edit web requests. A similar tool is [Charles HTTP proxy(Link opens in a new window)](http://www.charlesproxy.com/), which works on Windows, OS X, and Linux. Other tools include:

* [mitmproxy(Link opens in a new window)](http://mitmproxy.org/), a tool for OS X, Linux, and OpenBSD.
* The browser extensions mentioned under [Using browser-based options to test](https://help.tableau.com/current/api/rest_api/en-us/REST/rest_api_concepts_testing.htm#testing_using_browser-options) earlier in the topic. You can use extensions in Chrome and Firefox to examine requests that you make using tools like Postman and Advanced REST client.
* [Wireshark(Link opens in a new window)](https://www.wireshark.org/), a protocol analyzer that provides very detailed information about network traffic, including HTTP.

### **Examine log files**

Requests that you make to Tableau Server using the REST API result in log file entries on the server. You can examine the following log files to help troubleshoot REST API issues:

* *log-file-location*\httpd\access.*date-time-stamp*.log (for example, access.2015\_05\_11\_00\_00\_00.log).
* *log-file-location*\logs\vizportal\vizportal-*process-number*.log (for example vizportal-0.log)

For information about how to generate a snapshot of log files, see [Archive Log Files(Link opens in a new window)](https://help.tableau.com/current/server/en-us/help.htm#logs_create.htm). To learn more about log file locations, see [Server Log File Locations](https://help.tableau.com/current/server/en-us/logs_loc.htm).

# REST API Samples

[**https://github.com/tableau/rest-api-samples**](https://github.com/tableau/rest-api-samples)

# Get Started with Tableau Server on Windows

# https://help.tableau.com/current/server/en-us/get\_started\_server.htm

# Authentication and Authorization

[**https://help.tableau.com/current/blueprint/en-us/bp\_authentication\_authorization.htm**](https://help.tableau.com/current/blueprint/en-us/bp_authentication_authorization.htm)

# Network Communication

*Version: Current*

## **Tableau Server**

There are three main network interfaces in Tableau Server:

* Client to Tableau Server: The client can be a web browser, Tableau Mobile, Tableau Desktop, Tableau Prep Builder, or the tsm ([Windows](https://help.tableau.com/current/server/en-us/tsm.htm) | [Linux](https://help.tableau.com/current/server-linux/en-us/tsm.htm)) and tabcmd ([Windows](https://help.tableau.com/current/server/en-us/tabcmd.htm) | [Linux](https://help.tableau.com/current/server-linux/en-us/tabcmd.htm)) utilities.
* Tableau Server to your database(s): To refresh data extracts or handle live database connections, Tableau Server needs to communicate with your database(s).
* Server component communication: This applies to distributed deployments only.

You should configure SSL to ensure that access to Tableau Server is secure and that sensitive information passed between the server and Tableau clients is protected on your network. For more information, visit Network Security ([Windows](https://help.tableau.com/current/server/en-us/security_net.htm) | [Linux](https://help.tableau.com/current/server-linux/en-us/security_net.htm)) and Ports ([Windows](https://help.tableau.com/current/server/en-us/ports.htm) | [Linux](https://help.tableau.com/current/server-linux/en-us/ports.htm)).

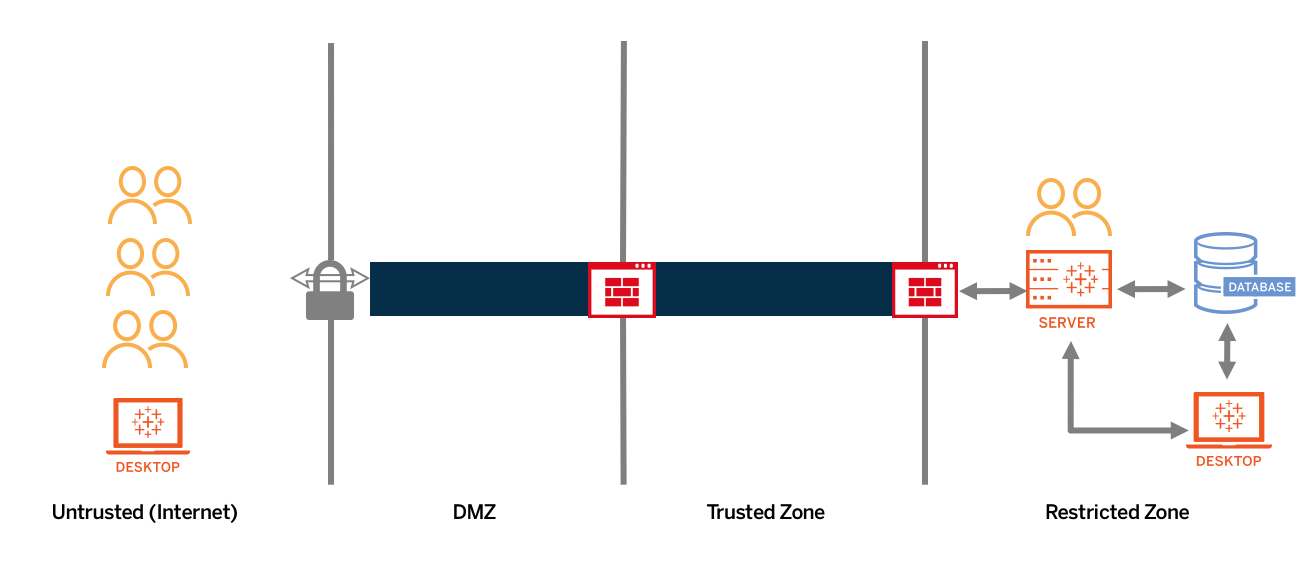
## **Tableau Online**

# Network Topologies

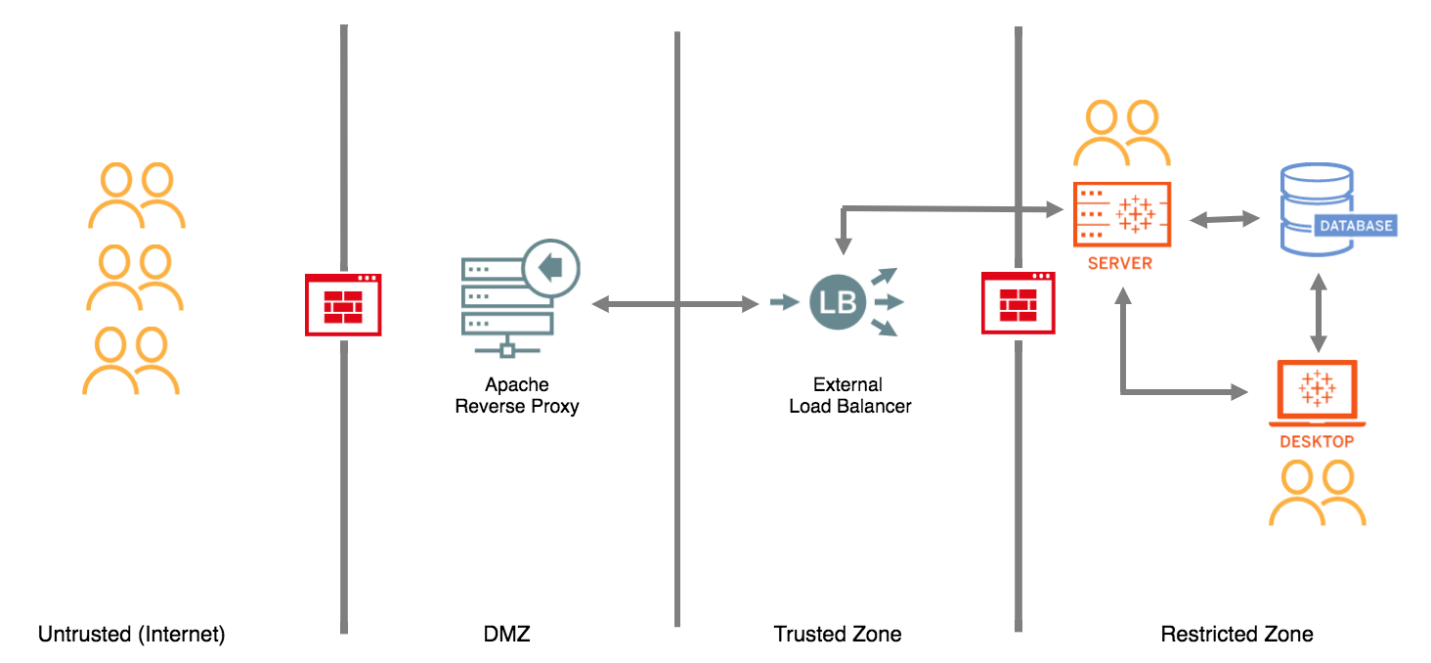
*Version: Current*

Users will be accessing Tableau from all over the place: their offices, on the road from their phones, from client sites in another country, and more. To ensure Tableau Server or Tableau Online is reachable no matter where the client is coming in from, you must determine the right network topology. Common network configurations are shown below:

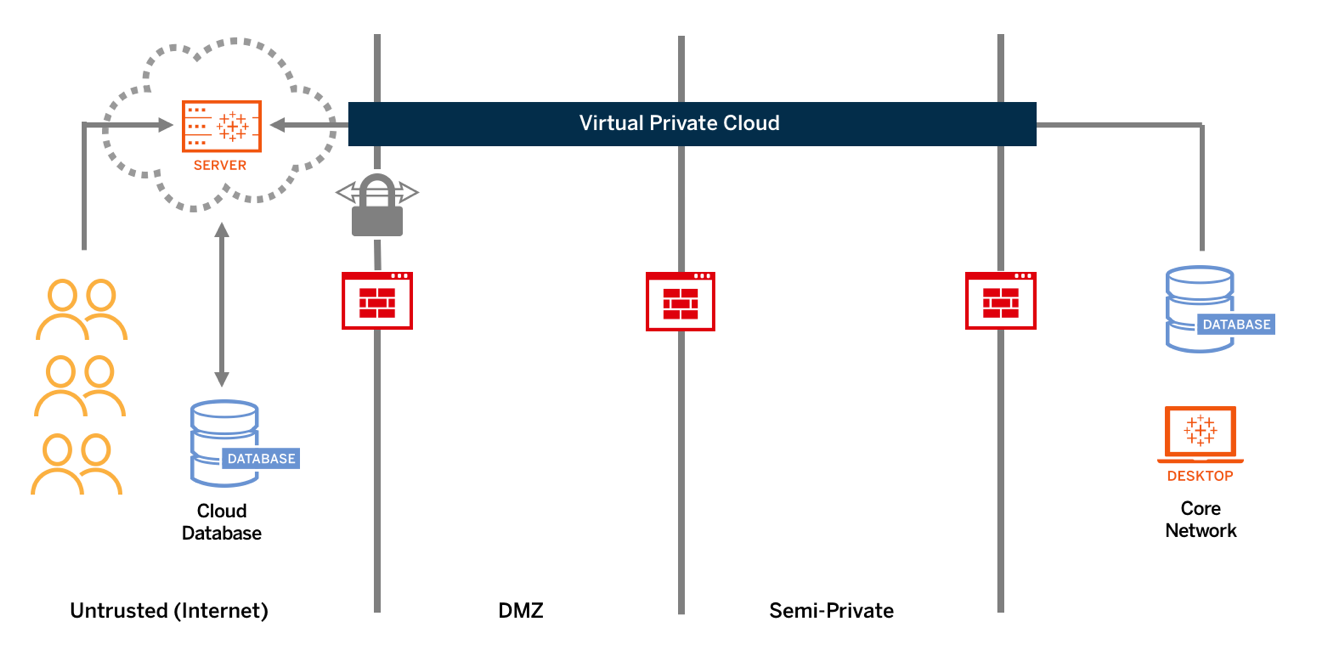
## **On-Premises with External VPN Access**



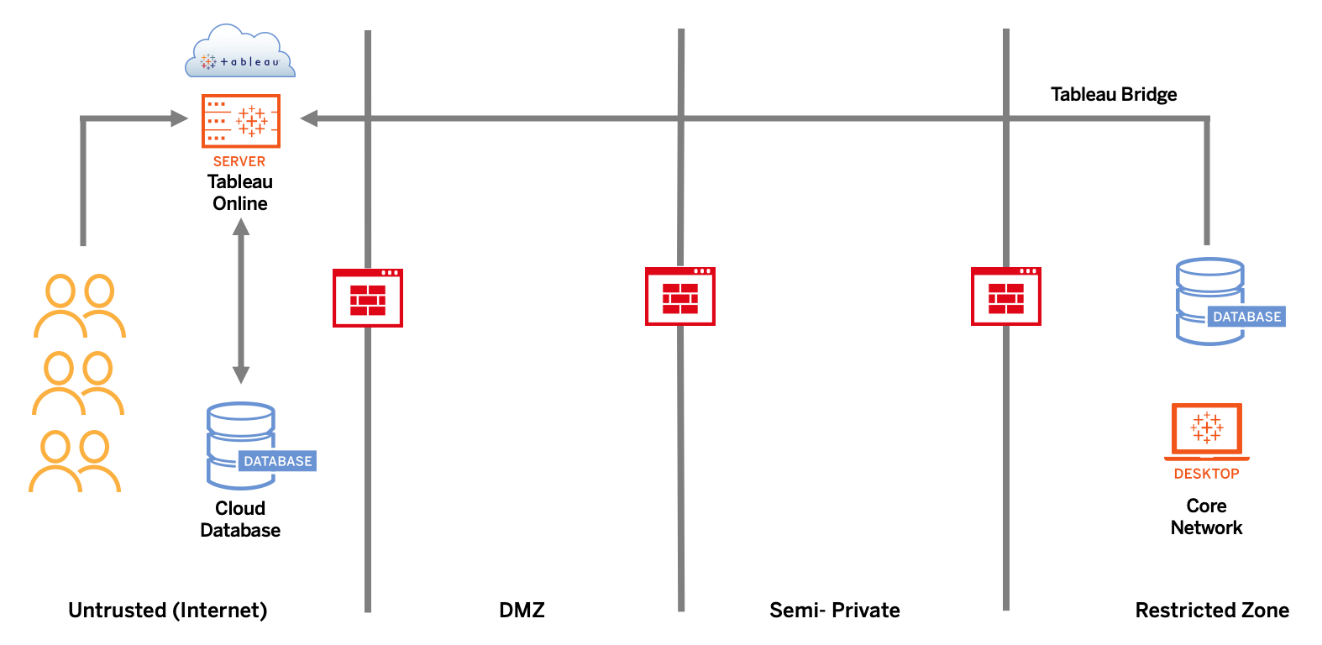
## **On-Premises with Reverse Proxy**

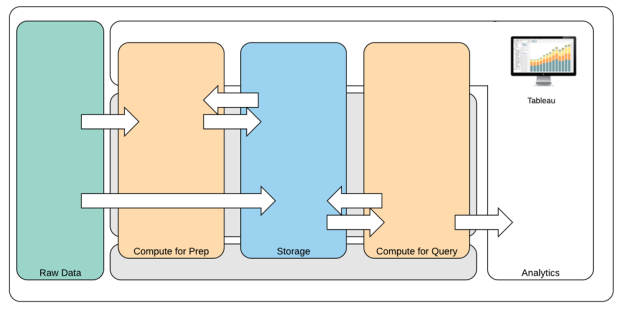


## **Public Cloud Deployment**



## **Tableau Online Deployment**





Modern Data Architecture

1. Raw Data: sources of data, for example, transactional data that is loaded into the data platform that often needs transforming in several ways: cleansing, inspection for PII, etc.
2. Compute for Prep: the processing of the raw data can require significant computational resources, so it's more than traditional ETL. Often Data Science apps will sit here. They actually can create new data with high value.
3. Storage: Modern data platforms are built on a principle of storing data because you never know how it might be used in the future. Increasingly, we store intermediate data and multiple versions and forms of the same data. Storage is therefore layered.
4. Compute for Query: the typical analytic database engine, including Hyper extracts, but also Hadoop, etc.
5. Analytics: Tableau sits in Analytics.

# Data Strategy

[**https://help.tableau.com/current/blueprint/en-us/bp\_data\_strategy.htm**](https://help.tableau.com/current/blueprint/en-us/bp_data_strategy.htm)

# Move Tableau to the Cloud

*Version: Current  
https://help.tableau.com/current/blueprint/en-us/bp\_move\_to\_cloud.htm*

From the beginning, Tableau has helped people see and understand data, no matter how they choose to do so. That philosophy extends to our customer’s deployment strategy. You can **host Tableau Server on-premises**, **host Tableau Server on a public cloud platform** (AWS, Google Cloud Platform, Microsoft Azure, or Alibaba), or **choose a fully-hosted offering, Tableau Online**. You are not locked into that choice once you begin your data journey. If your organization changes its data strategy, Tableau adapts with you.

As cloud adoption has increased, many customers have elected to move Tableau to the cloud. In this section we will highlight the major considerations you will have as you plan your transition to the cloud. These considerations are different depending on whether you would like to move Tableau Server to a public cloud or if you would like to transition from Tableau Server to Tableau Online.

Tableau Licensing

The main licensing consideration for customers who migrate to Tableau Online is that it utilizes Role-Based Subscription licensing. Tableau Server supports Role-Based Subscription licensing and legacy pricing models. If you are on a previous licensing model, please contact your Tableau account team to convert your licensing to Role-Based Subscription. For more information, see [Tableau License Management](https://help.tableau.com/current/blueprint/en-us/bp_license_management.htm).

# Security

[**https://www.tableau.com/security?\_ga=2.70255315.230086067.1620260467-1053483709.1619657013**](https://www.tableau.com/security?_ga=2.70255315.230086067.1620260467-1053483709.1619657013)

# Tableau Online Scalability

[**https://www.tableau.com/learn/whitepapers/tableau-online-scalability-overview-proof-points?\_ga=2.70255315.230086067.1620260467-1053483709.1619657013**](https://www.tableau.com/learn/whitepapers/tableau-online-scalability-overview-proof-points?_ga=2.70255315.230086067.1620260467-1053483709.1619657013)

# Connectivity with Bridge

[**https://help.tableau.com/current/online/en-us/to\_sync\_local\_data.htm**](https://help.tableau.com/current/online/en-us/to_sync_local_data.htm)

# Use Tableau Bridge

[**https://help.tableau.com/current/online/en-us/data\_fresh\_sync\_section.htm**](https://help.tableau.com/current/online/en-us/data_fresh_sync_section.htm)

# Hardware Platform

[**https://help.tableau.com/current/blueprint/en-us/bp\_hardware\_platform.htm#scalability**](https://help.tableau.com/current/blueprint/en-us/bp_hardware_platform.htm#scalability)

Modernize your analytics platform with Tableau on AWS

**https://www.tableau.com/solutions/Modern-Cloud-Analytics?\_ga=2.88485464.230086067.1620260467-1053483709.1619657013&\_fsi=qv7f7tfX**

# Tableau Desktop and Tableau Prep Builder Deployment

[**https://help.tableau.com/current/blueprint/en-us/bp\_client\_software.htm**](https://help.tableau.com/current/blueprint/en-us/bp_client_software.htm)

# Tableau Monitoring

[**https://help.tableau.com/current/blueprint/en-us/bp\_monitoring.htm**](https://help.tableau.com/current/blueprint/en-us/bp_monitoring.htm)

For Tableau Server, you can use a combination of hardware utilization monitoring, data from Tableau Server’s Repository, and external tools to understand and monitor the health of Tableau Server and how analytics are being consumed. Licensed as part of the [Tableau Server Management Add-on](https://help.tableau.com/current/server/en-us/itm_intro.htm), [**Resource Monitoring Tool**](https://help.tableau.com/current/server/en-us/rmt-intro.htm) is an agent-based monitoring application. Resource Monitoring Tool offers enhanced insights into the health and performance of a Tableau Server cluster by parsing and analyzing the logs generated by overall Server usage and combining that with monitoring of physical system usage (**CPU Usage, RAM, Disk I/O etc**.). In addition, standard enterprise monitoring platforms, such as Splunk, New Relic, or other agent-based utilities, can be used for monitoring. With all the data that Tableau and these solutions collect, Tableau Desktop should be used to analyze activity on your Tableau Server.

With viral adoption of Tableau in your organization, you will need to be responsive to the **increased workloads and proactively scale to avoid resource constraints**. On the other hand, assuming a correctly-sized hardware configuration is deployed initially, unmonitored hardware utilization and user workloads, inefficient workbooks, suboptimal data extract design, and peak-hour data refresh schedules can have a major impact on **server performance and user experience**, causing performance to degrade from the cumulative effect of the separate incidents.

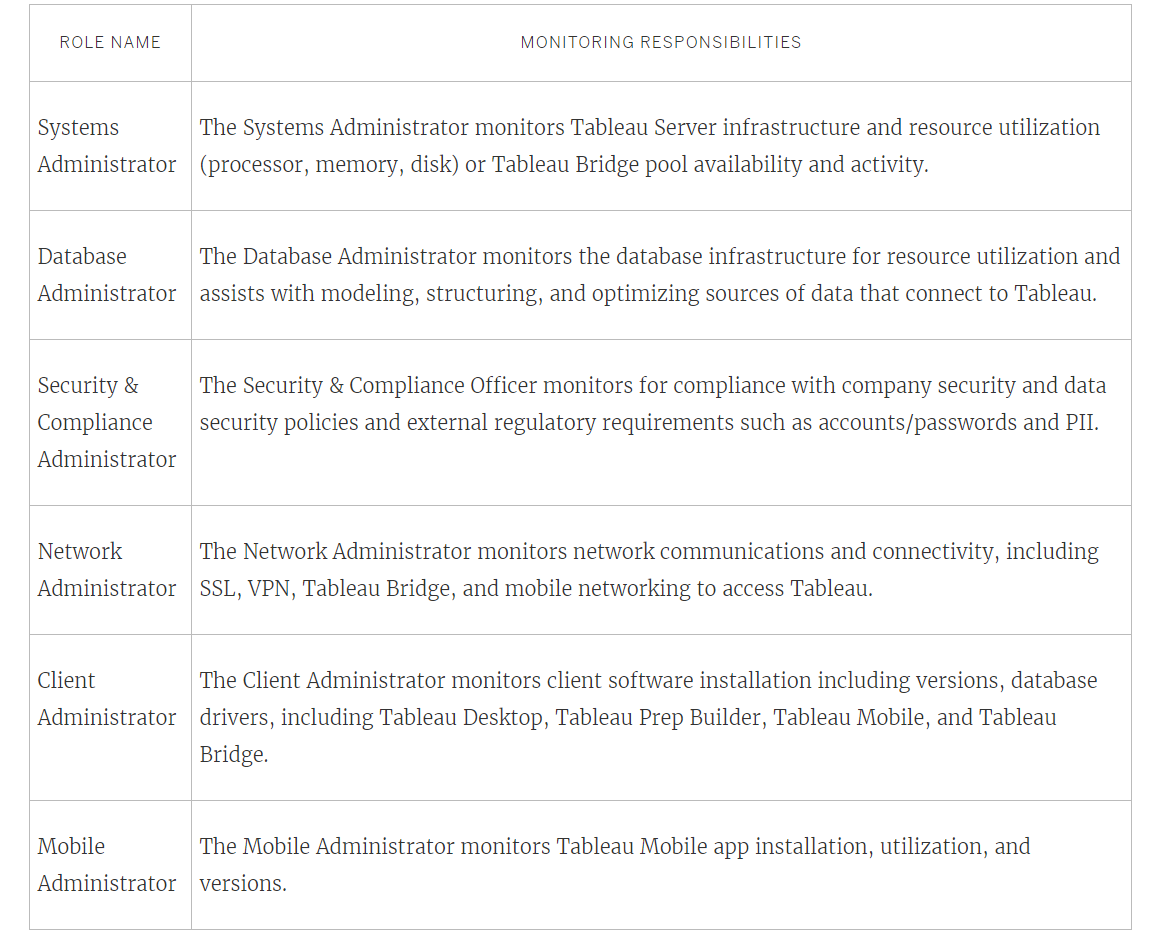
For Tableau Server monitoring and operations, systems administrators and Tableau Server Administrators should perform the following tasks:

* Set up tools to monitor hardware utilization and maintain sufficient headroom for future growth and peak usage spikes.
* Configure built-in alerts and incidents to notify you of events and thresholds.
* Use default administrative views to identify background task success/failure and job queueing, slow workbooks, disk usage, and license allocations.
* Create custom administrative views from repository data for deeper analytics on your deployment and set custom data-driven alerts for your own organization's policies and goals.
* Troubleshoot problems with Tableau Server logs.

**Because Tableau Online is a SaaS analytics platform, Tableau monitors the infrastructure and operations**; however, Tableau Online deployments still have monitoring tasks:

* Subscribe to [Tableau Trust](https://trust.tableau.com/) updates to get email, text, and Atom Feed or RSS Feed notifications when Tableau creates, updates or resolves an incident. At any time, you can visit the [Tableau Trust](https://trust.tableau.com/) page to view system status.
* **Monitor other systems** that are integrated with Tableau Online such as authentication, database, and network connectivity.
* Use **default administrative views** to identify background task success/failure and job queueing, slow workbooks, disk usage, and license allocations.
* **Create custom administrative views from Tableau Online Admin** Insights for deeper analytics on your deployment and set custom data-driven alerts for your own organization's policies and goals.

## https://help.tableau.com/current/blueprint/en-us/bp\_administrative\_roles\_responsibilities.htm



Hardware Monitoring

*https://help.tableau.com/current/blueprint/en-us/bp\_hardware\_monitoring.htm*

Any application is only as reliable and performant as the hardware that it runs on. It is important that the underlying infrastructure of your Tableau Server deployment be routinely monitored for capacity constraints to prevent overtaxing the system, whether it’s physical servers or virtual machines, on-premises or in the cloud. Systems administrators should monitor CPU, memory usage, storage I/O, storage space, and network bandwidth utilization.

Tableau Server

Beginning with 2019.3 and licensed as part of the [Tableau Server Management Add-on](https://help.tableau.com/current/server/en-us/itm_intro.htm), **the**[**Resource Monitoring Tool**](https://help.tableau.com/current/server/en-us/rmt-intro.htm) provides a comprehensive look at the health of Tableau Server using a **web user interface for all of its features**. The [Resource Monitoring **Tool Agent**](https://help.tableau.com/current/server/en-us/rmt-concepts.htm#agent-service) runs on each of the nodes in your Tableau cluster to monitor their hardware utilization, performance, and activity, which are collated by the [Resource Monitoring Tool Master Server](https://help.tableau.com/current/server/en-us/rmt-concepts.htm#master-service). Processor, Memory, Disk Queue, and Network are sampled using WMI several times per second to produce averages. An Environment Down event, which is logged as critical, is monitored at a 15 second polling interval by default and follows a three-strike rule. If the status is offline, or does not reply within 30 seconds, for three consecutive polling intervals then a critical incident is created. For more information, see [Performance](https://help.tableau.com/current/server/en-us/rmt-charts.htm#performance) and [Environment Down](https://help.tableau.com/current/server/en-us/rmt-incidents-environment-down.htm).

BEST PRACTICE RECOMMENDATION:

To maintain sufficient headroom and reduce the likelihood of resource constraints, calculate the following values based on your hardware specs and set Resource Monitoring Tool [Hardware incident thresholds](https://help.tableau.com/current/server/en-us/rmt-incidents-hardware.htm) to the following:

* CPU over 80% for 5 minutes
* Memory less than 25% available
* Disk less than 20% available

Depending on your enterprise monitoring approach, **hardware monitoring** can be done with an **agent-based enterprise monitoring service** or with [PerfMon](https://help.tableau.com/current/server/en-us/perf_collect_perfmon.htm) on **Windows** ([sample workbook](https://help.tableau.com/current/server/en-us/perf_analyze_sample_workbook.htm)).

**Linux** admins will use the **sysstat tools or vmstat** to collect data to be analyzed in a workbook similar to the Windows version.

When deploying on virtual platforms, such as VMware, HyperV, or Citrix, or in the public cloud, including [AWS](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring_ec2.html) (CloudWatch), [Microsoft Azure](https://docs.microsoft.com/en-us/azure/virtual-machines/windows/monitor) (Azure Portal), and [Google Cloud Platform](https://cloud.google.com/products/management/) (Google Stackdriver), these platforms have their own utilities for monitoring the health of their virtual machines. These tools usually have support for automatically monitoring your infrastructure and alerting you when your preset thresholds of 75-85% utilization are exceeded.

Alternatively, [TabMon](https://github.com/tableau/TabMon) is a freely available open source Tableau Server cluster-monitoring tool. TabMon provides structured output that is analyzed with Tableau Desktop. For more information, watch [A Tour of the TabMon Sample Workbook](https://www.youtube.com/watch?v=62fZ3py0ZDA) that shows how to monitor CPU, memory usage, storage I/O, storage space, and network bandwidth at the Tableau Server Process level. This information will help you understand when to scale Tableau Server. In the TabMon.config file, you should monitor every 300 seconds and keep data for 30-day intervals to manage the size of the database.

## **Tableau Online**

One of the benefits of Tableau Online is that it is managed and optimized by Tableau because it is a SaaS **analytics platform**. If you are connecting to on-premises data, Tableau Bridge availability and activity should be monitored. As a site administrator, you can monitor Tableau Bridge client in a couple of ways:

* [Monitor traffic to data sources](https://help.tableau.com/current/online/en-us/adminview_datasources.htm) with live connections using the built-in administrative view
* [Monitor client status on the site settings page](https://help.tableau.com/current/online/en-us/to_enable_bridge_live_connections.htm#understand-client-status)
* [Monitor Bridge extract performance](https://help.tableau.com/current/online/en-us/adminview_bridge_extract.htm) using the built-in administrative view
* Set up data source owners for extract refresh notifications

It’s important to note that **data source owners are the users** who receive emails when data sources fail. This is especially relevant in the centrally managed model, because the owners of the Bridge clients are the site admins, and it is commonly necessary to get them involved when issues occur and need troubleshooting. Because they will not be automatically notified, data source owners who are not the owner of the Bridge client generally set up automatic email forwarding to site admins to keep them up to date on issues.

As a data source publisher, if you believe an extract is taking an unusual amount of time, there are several things you can do to troubleshoot. An extract refresh through Bridge has two components: taking the extracts locally with Bridge, and then sending that extract to Online and having it become available. To isolate the potential cause of the issue, it may be prudent to take the same extract in Tableau Desktop and compare the extract duration with the time it takes for Bridge to take an extract. Any anomalies there should be reported to your Tableau contact for review and support.

If Tableau Server processes go down, the application state will be degraded or even fail depending on the process that goes down. During [installation and configuration](https://tableau.sharepoint.com/sites/deploymentframework/Shared%20Documents/Operational%20Guides/Architecture%2001%20-%20Deployment%20Guide.docx?web=1), Tableau Server was [configured with an SMTP email server](https://help.tableau.com/current/server/en-us/config_smtp.htm) to notify the specified email distribution list about [system alerts and failures](https://help.tableau.com/current/server/en-us/email.htm). With system alerts enabled, you will be notified by email of process up, down, and failover events and drive space constraints. See below for recommended settings.

## **Tableau Online**

Tableau Online Site Administrators should subscribe to [Tableau Trust](https://trust.tableau.com/) updates to get email, text, and Atom Feed or RSS Feed notifications when Tableau creates, updates or resolves an incident. At any time, you can visit the [Tableau Trust](https://trust.tableau.com/) page to view system status.

For extract refresh failures, the Tableau Online Site Administrator should enable Tableau Online to send email to the owner of a data source when its scheduled extract refresh does not complete successfully. The data source owner can then opt out individually in their account settings. For more information, see [Notify Owners When Extract Refreshes Fail](https://help.tableau.com/current/online/en-us/to_refresh_enable_emails.htm). For data sources that are refreshed through Tableau Bridge, notifications will vary. For more information, see [Manage Email Alerts for Bridge](https://help.tableau.com/current/online/en-us/to_bridge_alerts.htm).

# About Tableau Resource Monitoring Tool

[**https://help.tableau.com/current/server/en-us/rmt-intro.htm**](https://help.tableau.com/current/server/en-us/rmt-intro.htm)

Agent

Resource Monitoring Tool Agent runs as a background service on each of your Tableau servers. It must be installed on all the nodes in a Tableau Server installation.

The agent is a lightweight process that monitors the Tableau activity and general performance of the server it is installed on. It reports those details back to the Resource Monitoring Tool Master Server.

Master Server

The Resource Monitoring Tool Master Server hosts the web application that users interact with. It also does much of the background processing to collate and monitor the data from the agents.

What is Resource Monitoring Tool?

[**https://help.tableau.com/current/server/en-us/rmt-intro.htm**](https://help.tableau.com/current/server/en-us/rmt-intro.htm)

# Configuring Proxies for Tableau Server

[**https://help.tableau.com/current/server/en-us/proxy.htm**](https://help.tableau.com/current/server/en-us/proxy.htm)

## **Configure a forward proxy server**

To enable communication from Tableau Server to the internet, deploy Tableau Server behind a forward proxy server. When Tableau Server needs access to the internet, it doesn't send the request directly to the internet. Instead, it sends the request to the forward proxy, which in turn forwards the request. Forward proxies help administrators manage traffic out to the internet for tasks such as load balancing, blocking access to sites, etc.

If you use a forward proxy, you must configure the computers that run Tableau Server inside the network to send traffic to the forward proxy. Tableau Server doesn't support pass-through or manual proxy authentication.

How a reverse proxy works with Tableau Server

A reverse proxy is a server that receives requests from external (internet) clients and forwards them to Tableau Server. Why use a reverse proxy? The basic answer is security. A reverse proxy makes Tableau Server available to the internet without having to expose the individual IP address of that particular Tableau Server to the internet. A reverse proxy also acts as an authentication and pass-through device, so that no data is stored where people outside the company can get to it. This requirement can be important for organizations that are subject to various privacy regulations such as PCI, HIPAA, or SOX.

The following diagram illustrates the communication path when a client makes a request to Tableau Server that is configured to work with a reverse proxy server.

1. An external client initiates a connection to Tableau Server. The client uses the public URL that's been configured for the reverse proxy server, such as https://tableau.example.com. (The client doesn't know that it's accessing a reverse proxy.)
2. The reverse proxy maps that request in turn to a request to Tableau Server. The reverse proxy can be configured to authenticate the client (using SSL/TLS) as a precondition to passing the request to Tableau Server.
3. Tableau Server gets the request and sends its response to the reverse proxy.
4. The reverse proxy sends the content back to the client. As far as the client is concerned, it just had an interaction with Tableau Server, and has no way to know that the communication was mediated by the reverse proxy

# Install and Configure Tableau Server

[**https://help.tableau.com/current/server/en-us/install\_config\_top.htm**](https://help.tableau.com/current/server/en-us/install_config_top.htm)

# Automated Installation of Tableau Server

[**https://help.tableau.com/current/server/en-us/automated\_install\_windows.htm**](https://help.tableau.com/current/server/en-us/automated_install_windows.htm)

# Distributed and High Availability Tableau Server Installations

[**https://help.tableau.com/current/server/en-us/distrib\_ha.htm**](https://help.tableau.com/current/server/en-us/distrib_ha.htm)

* **Single-node installation**—This type of installation is reasonable for testing, running trials, and for environments that can handle occasional downtime and system availability due to lack of redundancy. All server processes are running on a single machine. There is less redundancy and fewer safeguards in the event of a problem with one of the server processes. You also need to make sure the computer you install Tableau Server on has adequate resources to handle the processes and the demands of users and data.
* **Distributed installation**—This type of installation is also called a multi-node installation and requires multiple computers so you can install and run server processes on those distributed nodes. Spreading the server processes out over multiple nodes can extend the reliability and efficiency of Tableau Server by providing redundancy and additional computing power. With the right configuration, a distributed installation can also provide you with automatic repository failover. For more information on failover, see [Repository Failover](https://help.tableau.com/current/server/en-us/distrib_ha_failover.htm).
* **Highly available (HA) installation**—An HA installation of Tableau Server is a special type of multi-node installation with a minimum of three nodes and multiple instances of key processes (the Repository, File Store/Data Engine (Hyper), Coordination Service, and Client File Service) on different computers. With an HA installation, there is built-in redundancy of those key processes, including multiple File Stores, and automatic Repository failover. The goal is to minimize system downtime by eliminating single points of failure, and enabling detection of failures with failover where possible.

# Add a Load Balancer

[**https://help.tableau.com/current/server/en-us/distrib\_lb.htm**](https://help.tableau.com/current/server/en-us/distrib_lb.htm)

# Get Started with Tableau Server on Windows

[**https://help.tableau.com/current/server/en-us/get\_started\_server.htm**](https://help.tableau.com/current/server/en-us/get_started_server.htm)

**Automation**

[**https://www.tableau.com/developer/automation**](https://www.tableau.com/developer/automation)