



Axcient

Architecting a High Availability Private
Cloud Environment

axcient.com

Contents

Contents	2
Introduction	5
Architectural Overview	5
Scaling Parameters Summary and Examples	7
Basic Server Requirements.....	9
Advanced Server Requirements	10
Anchor Server Requirements.....	10
Apache Server Requirements	10
PostgreSQL Portal DB Server Requirements	10
PostgreSQL Root DB Server Requirements.....	11
Bandwidth Requirements.....	12
Storage Requirements	12
Load Balancing and Networking Recommendations.....	14
Setting Up a High Availability Configuration	15
Configuration Examples.....	15
Requirements	16
PostgreSQL Performance Optimization.....	16
Getting Started Checklist.....	18
Configuration Instructions for minimal setup	19
Step 1: Install Anchor and PostgreSQL on Machine 1.....	19
Step 2: Install Anchor and Apache on Machine 2	25
Step 3: Install Anchor on Machine 3	26
Option 1: Configuring a Locally Mounted Storage Drive	28
Configuration Instructions.....	28
Step 1: Map Drives	28
Step 2: Register Anchor	31

Step 3: Modify Anchor Configuration Files	33
Option 2: Configuring a NAS, SAN, or Other Storage Device	36
Configuration Instructions.....	36
Step 1: Create Local Admin Accounts on Each Machine.....	36
Step 2: Register Anchor	36
Step 3: Copy Metadata	38
Step 4: Change Services to Run as Local Admin	38
Step 5: Modify Anchor Configuration Files.....	38
Step 6: Update PostgreSQL with the UNC Path	40
Configuring Backups	44
Reviewing Mappings for the Load Balancer or DNS Server	45
Configuring Dual Hostname Settings.....	46
Configuration Instructions.....	47
Step 1: Configure a Second Domain or Subdomain	47
(Optional) Step 1a: Configure an Additional IP Address.....	47
(Optional) Step 1b: Stop the Apache and Anchor Services	50
Step 2: Point the New Domain or Subdomain to the new IP Address.....	51
Step 3: Configure the Port Address Translation (PAT) Settings	51
Step 4: Configure the SSL Certificates.....	51
Step 5: Update the Apache Configuration File.....	52
Step 6: Specify the App Server Hostname within the Web Portal	54
Setting up PostgreSQL connection pooling.....	56
Installation of PGBouncer.....	56
Configuration of PGBouncer.....	57
Configuration of Anchor servers to use connection pooler	58
Scaling the number of Apache servers.....	58
Deploying the new Apache servers	58
Setting up common shared static assets storage and shared services.....	59

Step 1: Create a domain or regular user with permissions to run Apache service.....	59
Step 2: Create a networking share for assets storage	60
Step 3: Move assets to shared storage	60
Step 4: Verify and configure Celery and Redis services	61
Step 5: Configure web application on all Apache servers to use common resources	63
Step 6: Start Apache servers and add them to load balancing setup	65
PostgreSQL and Anchor connection limits	65
Relevant configuration parameters	66
Calculating the required limits to set for PostgreSQL	67
Finding Additional Support	70

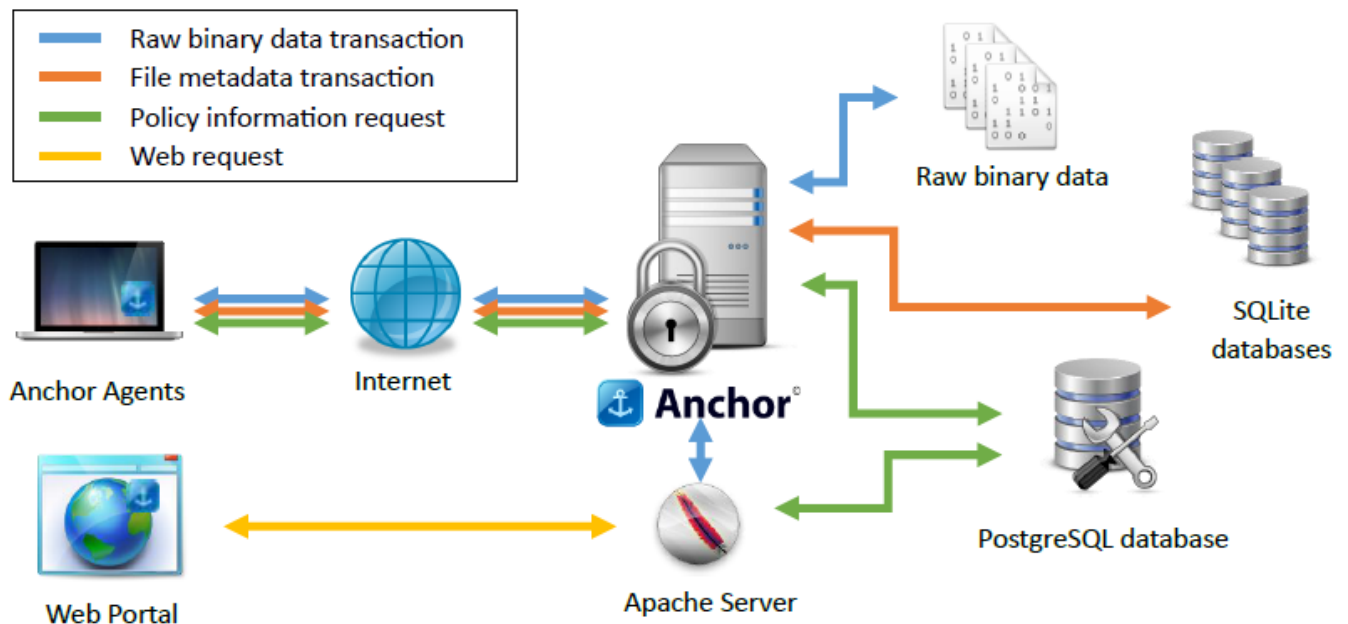
Introduction

x360Sync (formerly known as Anchor) is a software platform that enables real-time file synchronization, sharing, collaboration, and backup for businesses. With a focus on security, reliability, control, and integration, Anchor provides IT professionals with a solution to the universal access, sharing, and file-recovery problems that plague organizations.

The platform is comprised of two services: server and agent. The server service can be hosted within a company’s internal infrastructure (private cloud) or can be provided by Anchor’s hosted cloud infrastructure (SaaS). This guide will help you plan and implement an Anchor private cloud environment according to established guidelines and best practices.

Architectural Overview

The Anchor software architecture is comprised of various components:



- **Anchor Agents and Apps**—software applications for Windows and OSX, and mobile apps for iOS and Android. These agents and apps communicate with both the Apache Server and Anchor Server.
- **Web Portal**—an HTML5/Javascript frontend that communicates with the API on the Apache Server.
- **Anchor Servers**—one or more virtual or physical servers running the proprietary Anchor backend server software on Windows. Each agent will keep one persistent

connection open to one Anchor Server. If multiple servers are configured, a cluster is formed, allowing agents to connect to any cluster member to service any request, and allowing instant yet efficient notification of file changes to any connected agents. The Anchor server is dependent on the PostgreSQL database, SQLite databases, and the raw binary data stores.

- **Apache Server**—a virtual or physical server running Apache and the proprietary Anchor web backend application on Windows and is configured to use your custom SSL certificate. The Apache server depends upon the Anchor servers, PostgreSQL, and the SQLite databases.
- **PostgreSQL DB**—stores policy and account information (Portal database). Latest versions of Anchor Server also use it by default to store root information (Root databases) like list of files and folders in share, revisions of files, etc., instead of SQLite databases. We recommend a high availability PostgreSQL setup with master/slave replication, as the Anchor service will not operate if PostgreSQL is down. Also, for deployments which expected to be used by 2000+ clients, it's recommended to run separate PostgreSQL servers for Portal and Root databases to not have slowdowns on the Portal database due to high load on the Root databases. *You must back up the PostgreSQL databases on a regular basis.*
- **SQLite databases** (root stores)—legacy way of storing information about each revision of a synchronized file, with one SQLite database per top-level folder (root) synchronized. One or multiple mount points are supported. For Anchor private cloud deployments across more than one server, the SQLite databases must be accessible on a network addressable filesystem that can be accessed concurrently by multiple systems; this filesystem must also support proper file locking, such as CIFS or NFSv4. This way of storing databases is legacy, kept for backward compatibility, and not recommended to be used for new deployments. *You must back up the SQLite databases on a regular basis.*
- **Raw binary data** (file stores)—one or more filesystem mount points that will store the actual encrypted binary data for each file revision. For Anchor private cloud deployments across more than one server, the file stores must be accessible on a network filesystem, such as CIFS or NFSv4. A subdirectory structure will automatically be created so that any individual directory will only have a maximum of a few thousand files in it. *You must back up the file stores on a regular basis.*

Only the Anchor servers and Apache server needs to be exposed to the Internet through your firewall/router or load balancer. All external communication is over well-defined TCP ports. Public DNS entries are needed for the Anchor and Apache servers. In many cases, round robin DNS with a short TTL is sufficient for load balancing, or any load balancer that supports TCP (for example, HAProxy, F5) but keep in mind what sticky client sessions are recommended for optimal performance.

This document gives general guidelines for infrastructure, based on observed average resource requirements for at-scale systems. Your actual resource requirements might vary based on usage patterns. It is typically observed that for every 1000 users provisioned in the system, there are 700 active connections.

Scaling Parameters Summary and Examples

For every 1400 provisioned users, you will need the following resources (it is usually observed that for every 1400 provisioned users, there are 1000 active connections).

Resource	1400 Users
Anchor Server vCPU	2.5
Anchor Server vRAM (GB)	5
Apache Server vCPU	0.25
Apache Server vRAM (GB)	1
PostgreSQL vCPU for Portal DB server	0.25
PostgreSQL vRAM (GB) for Portal DB server	0.5
PostgreSQL vCPU for Root DB server	2
PostgreSQL vRAM (GB) for Root DB server	6
Bandwidth (Mbit/sec)	35

For every 1000 provisioned users, you will need the following storage resources:

Resource	1,000 Users
PostgreSQL storage (GB) <i>SSD storage required</i>	5
Root stores storage (GB) <i>SSD storage required</i>	4
Root stores avg read IOPS	150
Root stores max read IOPS (burst)	300
Root stores avg write IOPS	80
Root stores max write IOPS (burst)	200
File stores storage (GB) Storage averages will vary greatly per use case; 7200 rpm storage is sufficient	15,000 (15 TB)

The following table estimates the resources required to support a given number of *provisioned* users:

Resource	1,000 Users	5,000 Users	10,000 Users	25,000 Users	50,000 Users	100,000 Users
Anchor Server vCPU	1.8	8.8	17.5	43.8	87.5	175.0
Anchor Server vRAM (GB)	3.5	17.5	35.0	87.5	175.0	350.0
Apache Server vCPU	0.2	0.9	1.8	4.4	8.8	17.5
Apache Server vRAM (GB)	0.7	3.5	7.0	17.5	35.0	70.0
PostgreSQL vCPU for Portal DB server	0.2	0.9	1.8	4.4	8.8	17.5

PostgreSQL vRAM (GB) for Portal DB server	0.4	1.8	3.5	8.8	17.5	35.0
PostgreSQL vCPU for Root DB server	1.6	8	16	32	64	80
PostgreSQL vRAM (GB) for Root DB server	5	10	20	50	100	200
PostgreSQL storage (GB)	5	25	50	125	250	500
Root stores storage (GB)	4	20	40	100	200	400
Root stores avg read IOPS	150	750	1,500	3,750	7,500	15,000
Root stores max read IOPS	300	1,500	3,000	7,500	15,000	30,000
Root stores avg write IOPS	80	400	800	2,000	4,000	8,000
Root stores max write IOPS	200	1,000	2,000	5,000	10,000	20,000
File stores storage (GB)	15,000	75,000	150,000	375,000	750,000	1,500,000
WAN bandwidth (Mbit/sec)	25	123	245	613	1,225	2,450
WAN outbound GB/month	3,500	17,500	35,000	87,500	175,000	350,000
WAN inbound GB/month	1,750	8,750	17,500	43,750	87,500	175,000

In addition to the estimated resources above, it is important to consider high availability requirements—as well as the scalability of the network file storage—for your root stores and file stores.

Basic Server Requirements

By default, the Anchor Server Installer installs a single Anchor, Apache, and database server on one machine (physical or virtual). The basic recommended server requirements for this

type of environment can be found in the [Anchor Architecture Overview](#) Knowledgebase article. The recommended system requirements in this article are known to support 5,000 simultaneous connections. A single server can scale up to much higher simultaneous connection counts, as per guidelines given in this document. Additionally, the Anchor servers can scale out to as many servers as needed to meet the load.

Advanced Server Requirements

For an advanced rollout—intended to support a high number of users—it is recommended that you utilize a high availability configuration and scale out as needed.

Anchor Server Requirements

- Any physical or virtual x64 server that is compatible with Windows Server
- Requires at least 4 CPU cores and 6 GB of RAM
- Gigabit NICs are required; 10Gbit is recommended
- 2.5 vCPUs (CPU cores) and 5 GB vRAM per 1000 active connections (for example, a dual-socket Xeon E5-2670v3 system with 64GB RAM would support 10,000 active connections)
- For high availability, start with two servers and scale out as needed (for example, an environment that intends to support 300,000 active connections might use 30 Anchor servers)

Apache Server Requirements

- Any physical or virtual x86 server that is compatible with Windows Server
- Requires at least 4 CPU cores and 6 GB of RAM
- 0.25 vCPUs and 1 GB vRAM per 1000 active connections

PostgreSQL Portal DB Server Requirements

- Any physical or virtual x86 server that is compatible with Windows Server or Debian amd64 Linux
- Requires at least 4 CPU cores and 6 GB of RAM
- 0.25 vCPUs and 0.5 GB vRAM per 1000 active connections
- 5 GB storage per 1000 users (fast NVMe SSD storage is required)
- Two PostgreSQL servers (master and slave) running in a cluster, each having an independent IP address; an additional cluster IP address will only be used by the master node. If the cluster fails over, then the slave becomes the master and takes

over the cluster IP address. The Anchor app and Web app utilize the cluster IP when talking to PostgreSQL.

PostgreSQL Root DB Server Requirements

- Any physical or virtual x86 server that is compatible with Windows Server or Debian amd64 Linux
- Requires at least 4 CPU cores and 6 GB of RAM
- 1.6 vCPUs and 5 GB vRAM per 1000 active connections
- 5 GB storage per 1000 users (fast NVMe SSD storage is required)
- Two PostgreSQL servers (master and slave) running in a cluster, each having an independent IP address; an additional cluster IP address will only be used by the master node. If the cluster fails over, then the slave becomes the master and takes over the cluster IP address. The Anchor app and Web app utilize the cluster IP when talking to PostgreSQL.

For both PostgreSQL Portal DB Server and PostgreSQL Root DB Server we recommend using PostgreSQL version 14 which was tested to work properly and efficiently on our SaaS deployment.

For both PostgreSQL Portal DB Server and PostgreSQL Root DB Server connection pooling is recommended, required for deployments with 4+ Anchor servers. We recommend using PGBouncer running on a separate node for connection pooling setup: <https://www.pgbouncer.org>



Note: You must *frequently* back up your PostgreSQL databases.

For more information on PostgreSQL high availability, please reference the following articles:

- [PostgreSQL High Availability](#)
- [DiskSpd Tool](#)
- [Failover Clustering Hardware Requirements](#)
- [Deploying a Hyper-V Cluster](#)



Note: by default, any PostgreSQL install is NOT configured for optimal performance, and it's absolutely required to tune up the PostgreSQL configuration to be able to use system resources efficiently. More information on that you can find in "Setting Up a High Availability Configuration" section.

Bandwidth Requirements

Bandwidth usage will vary based on synchronization and collaboration patterns, but typically follows the scaling parameters as described below.



Note: The numbers given here assume that LAN Sync is turned off, and that each agent endpoint is in a different LAN network (worst case scenario). Please note that Anchor's LAN Sync feature should be turned on to allow multiple agents in the same LAN to pull changes from each other, rather than across the Internet from the Anchor servers. Turning on LAN Sync will significantly reduce bandwidth requirements.

Each month, every active connection will use about 5 GB of outbound bandwidth and 2.5 GB of inbound bandwidth. We estimate that for every 1000 active connections, you will need approximately 35 Mbit/sec of available Internet bandwidth across your edge router/firewall.

Internet network traffic between cluster members is relatively minimal, except for any network traffic relating to the network-accessible root stores and file stores, which can be significant (see storage requirements section for details).

Storage Requirements

In an Anchor private cloud system with multiple Anchor servers, you must use network addressable storage mount points for both root stores and file stores that support file locking and concurrent access (e.g., CIFS or NFSv4).



Note: The Apache server only needs *read* access to root stores and does not need access to file stores. This helps minimize file locking traffic caused by Apache servers accessing the root stores.

Root Store Recommendations

- SSD storage is *required* for proper performance for systems with 1000 users or more
- The system can distribute root store DBs across multiple root store mount points
- Characteristics per 1000 provisioned users:
 - Average SQLite DB file is 1.5 MB
 - 4 GB data
 - 150 read IOPS (average), 300 read IOPS (burst)
 - 80 write IOPS (average), 200 write IOPS (burst)
- Any POSIX network filesystem with file locking is supported (for example, CIFS, NFSv4)
- Possible systems for small or non-HA setups:
 - Solaris/Linux/BSD/FreeNAS ZFS over SSDs, exported via NFSv4
 - Windows hardware RAID6 over SSDs, exported via CIFS
- Possible systems for larger or HA setups:
 - Scale-out NAS/SAN that expose filesystem storage over CIFS or NFSv4
 - Software-defined storage, such as Scality



Note: SQLite test scripts can be requested to ensure your desired storage is compatible.



Note: You must *frequently* back up your root store data.

File Store Recommendations

- 7200 rpm storage is sufficient, but SSD storage is recommended for systems with 4000+ clients
- The system can distribute file data across multiple file store mount points
- Characteristics per 1000 provisioned users:
 - Average file is 0.8 MB
 - 15,000 GB (15 TB) data and 18,750 files per 1000 users
- RAID6 (or equivalent high levels of redundancy) and a storage subsystem with proper on-demand and periodic integrity checking and self-healing for the data

- Any POSIX network filesystem is supported (file locking is not required)
 - Typically, network storage systems are used that can provide CIFS or NFSv4 storage at scale



Note: You must *frequently* back up your file store data.

Load Balancing and Networking Recommendations

- Both the Anchor and Apache servers use the TLS (SSL) protocol to encrypt transmissions
- An SSL/TLS offloader is *not* required
- Edge routing and firewall equipment must support the number of concurrent TCP connections matching the number of users; it must also support at least 40 Mbit/sec of throughput per 1000 users
- Load Balancer should use sticky sessions for optimal performance



Note: Edge routing and firewall equipment should be configured in high availability clusters to ensure proper uptime of the service.

Setting Up a High Availability Configuration

For advanced private cloud environments—intended to support a high number of users—it is recommended that you configure a high availability (HA) environment, where the Apache servers and PostgreSQL database servers are installed on separate machines, and Anchor is replicated on multiple machines. Ultimately, this eliminates a single-point of failure at the Anchor level, and allows for distribution of load across multiple Anchor servers.



Note: These instructions will help you set up a high availability configuration for the Anchor server only. While this process allows you to place Apache or PostgreSQL on the server of your choice within the high availability cluster, it does not replicate the Apache or PostgreSQL servers. The system will only recognize one instance of the Apache server and one instance of the PostgreSQL server. For instructions on how to increase the number of Apache servers, refer to section “Scaling up the number of Apache servers”.

For more information on setting up a windows cluster in VMware, please reference:

- [VMware vSphere High Availability](#)
- [Creating a vSphere HA Cluster](#)

For more information on setting up fault tolerance with Native VMware, please reference:

- [Setup for Failover Clustering and Microsoft Cluster Service](#)
 - [Microsoft Clustering on VMWare vSphere](#)
-

Configuration Examples

High availability can be configured in various ways.

For example, when using two machines, you can configure the high availability cluster as follows:

- Machine 1: Anchor server, Apache server, and PostgreSQL server
- Machine 2: Anchor server

When using three or more machines, you can configure the high availability cluster as follows:

- Machine 1: Anchor server and PostgreSQL

- Machine 2: Anchor server and Apache server
- Machine 3 (or more): Anchor server

Optimal HA cluster:

- Machine 1: Dedicated PostgreSQL server for Portal DB
- Machine 2: Dedicated PostgreSQL server for Root DB
- Machine 3: Miscellaneous services server with PGBouncer and Redis
- Machine 4: Load balancer for Apache nodes (Haproxy recommended)
- Machine 4: Anchor server
- Machine 5 (or more): Anchor server
- Machine 6: Apache server
- Machine 7 (or more): Apache server

Requirements

The following items are required when configuring an environment for high availability:

- Two or more servers that meet system requirements as described above
- Load balancer or Round-robin DNS (with short 5-minute TTL), configured to use sticky sessions
- Root store, root database server and file store network storage mount points that can be accessed by all machines



Note: Storage can be permanently mounted on one server, and mapped to the other servers. NAS and SAN devices are also supported. This guide provides instructions for configuring both storage options.

PostgreSQL Performance Optimization

By default, any PostgreSQL install is NOT configured for optimal performance (instead it's configured for minimal resource usage), and it's absolutely critical to tune up the PostgreSQL configuration to be able to use system resources efficiently for optimal performance if you're planning to use your deployment for large numbers of clients.

Most of the runtime parameters mentioned below, can only be set on PostgreSQL server start, so it makes sense to set these permanently in PostgreSQL server configuration on deployment. On Windows, this file is typically located at

C:\Program Files\PostgreSQL\<<version>\data\postgresql.conf

Main parameters we're interested in are the following:

max_connections - Determines the maximum number of concurrent connections to the database server. The default is typically 100 connections. Every Anchor server typically keeps around 40-50 simultaneous connections to the database server all the time, so in environment where you have 2+ Anchor servers, default won't be enough, and you will get slowdowns related to Anchor servers competing for available connection slots on PostgreSQL servers. We recommend setting this parameter to a minimum of 150 connections for PostgreSQL server for Portal DB and to minimum of 500 connections for PostgreSQL server for Root DB. To evade tuning this parameter up in the future and optimize the connections availability and RAM usage, if your deployment required more than two Anchor servers, we strongly recommend setting up PostgreSQL connection pooling using [PGBouncer](#), as every connection requires shared memory allocation, and if your PostgreSQL instance experiences a surge of simultaneous connections, it can exhaust the system's shared memory without pooling in place. More information on fine-tuning the connection limits can be found in the section "PostgreSQL and Anchor connection limits" in the end of this manual.

shared_buffers – This one is extremely important. It sets the amount of memory the database server uses for shared memory buffers which directly affects the overall server performance, and by default, it's extremely small (typically set to 128MB). For proper performance, reasonable starting value for `shared_buffers` should be around 25% of the memory in your system. So, for the PostgreSQL server with 10GB of RAM, adequate value to set would be 2048MB.

effective_cache_size – This parameter informs the PostgreSQL runtime about the amounts of cache memory available on the system in both disk and RAM cache which PostgreSQL runtime can use for operations. The size of this parameter directly affects how aggressively PostgreSQL will try to optimize the operations when running the queries. This parameter has no effect on the size of shared memory allocated by PostgreSQL, nor does it reserve kernel disk cache; it is used only for estimation purposes. Default is set to 2 or 4 GBs depending on exact PostgreSQL distribution, we recommend keeping it around 25% of the memory in your system if your PostgreSQL is running on a dedicated server.

max_worker_processes - Sets the maximum number of background processes that the cluster can support, the default is 8. If your PostgreSQL server has more than 8 CPU cores it would benefit greatly from increasing this parameter. Increase to `max_parallel_workers` + other workers, such as workers for logical replication and custom background workers, but not more than number of cores available in the system.

max_parallel_workers - Sets the maximum number of workers that the cluster can support for parallel operations (like running queries in parallel specifically). The default value is 8. Similarly to **max_worker_processes**, a good default is to set max_parallel_workers equal to number of CPU cores available in the system.

work_mem - Sets the base maximum amount of memory to be used by a query operation (such as a sort or hash table) before writing to temporary disk files. Default is 4MB. Writing to disk is an expensive operation which takes a lot of time so it makes sense to keep as much data in memory as possible for most of the cases. A good idea for this value would be to set it according to formula

$$\text{work_mem} = (25 \% \text{ of RAM in MB}) / \text{max_connections}$$

For example, for PostgreSQL server with 10GB or RAM and 150 connections limit for **max_connections**, adequate value for work_mem would be 16MB. Be careful not to set it too high as it's easy to run out of memory this way.



Note: SQL servers require a lot of RAM to work efficiently, so the best approach for HA deployments is to keep PostgreSQL servers on a dedicated machines which have no other software or services running so that PostgreSQL won't have to compete for resources. We recommend running PostgreSQL servers on Debian Linux server distribution which has a minimal system core and nothing else. This way all the system resources on the server would be available for PostgreSQL alone.

To be able to estimate the actual max RAM which PostgreSQL will likely use, use the following formula: $(\text{temp_buffers} + \text{work_mem}) * \text{max_connections} + \text{shared_buffers}$

Getting Started Checklist

It is assumed that the following tasks have been completed:

- Servers are not members of a domain
- IIS or additional web server software has been removed from each machine
- Ports 443, 80, and 510 are all available (dual hostnames will later be configured to remove the need for port 510)
- Data center infrastructure and networking components are installed

- Database servers are installed, configured and running if you are going to use dedicated SQL servers for deployment
- The following access rights have been acquired:
 - Access to the load balancer/DNS server
 - Administrative privileges on all machines
- SSL certificates are installed and configured
- IP addresses of each server have been recorded for easy reference

Configuration Instructions for minimal setup

Step 1: Install Anchor and PostgreSQL on Machine 1

After you prepare for the high availability configuration, you can install the Anchor and PostgreSQL servers on machine 1.



Note: Anchor must be installed on the PostgreSQL server first

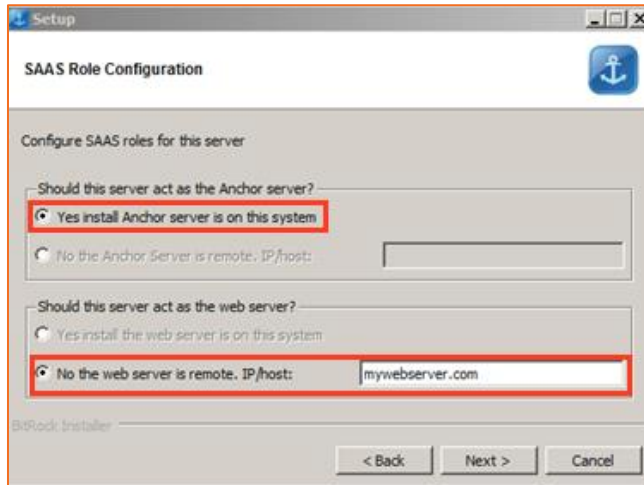
1. Log into machine 1 (the Anchor and PostgreSQL server) as a local admin.
2. On the server, download the latest Anchor installation file.
3. Open an elevated Command Prompt window. The Command Prompt window displays.
4. In the Command Prompt window, run the Anchor Server executable with the `--saas 1` argument, which will allow you to select specific components to install during the installation process.

```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\Administrator\Desktop>AnchorServer.exe --saas 1
```

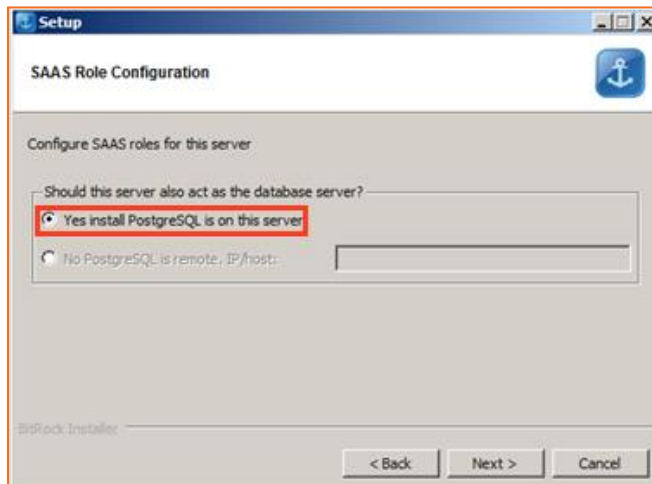
The *Anchor Setup Wizard* will launch.

5. In the *Anchor Setup Wizard*, click the series of **Next** buttons to proceed through the setup process, making sure to pause at the *SAAS Role Configuration* screen.
6. In the *SAAS Role Configuration* screen, select the following options:
 - a. Select the Yes install Anchor server on this system radio button.
 - b. Select the No the web server is remote radio button.
 - c. Enter the web server **IP address** or **hostname**.

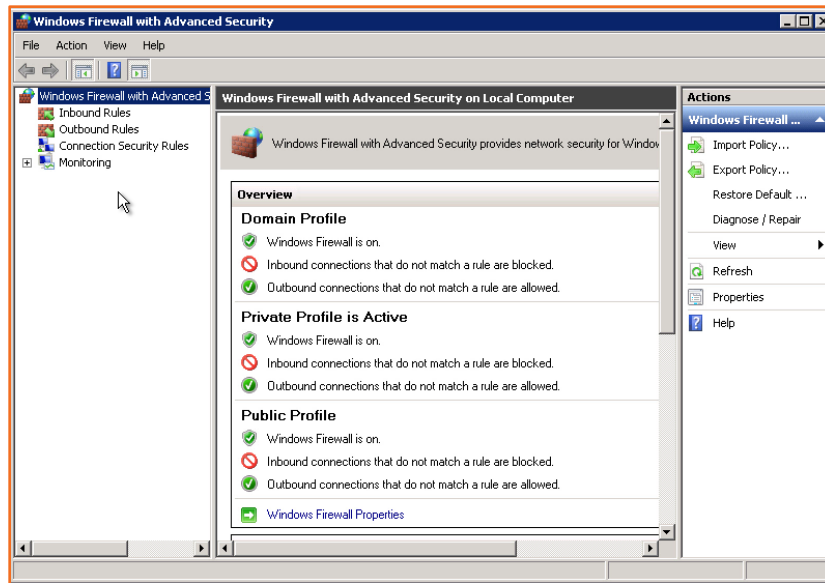
- d. Click the **Next** button when you are finished.



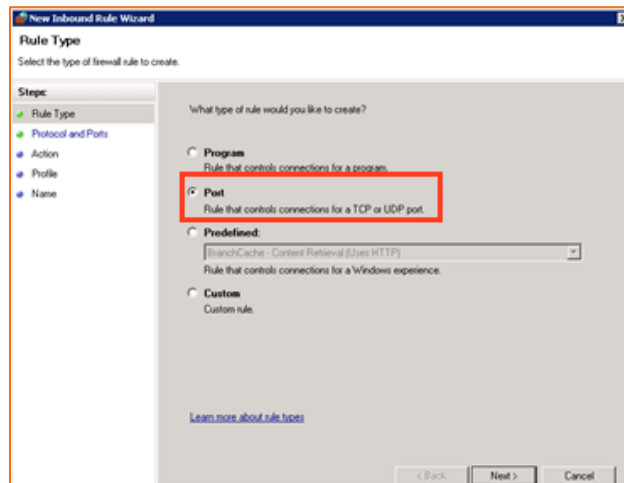
7. In the next *SaaS Role Configuration* screen, click the **Yes install PostgreSQL is on this server** radio button, and then click the **Next** button.



8. When the *Anchor Setup Wizard* completes, click the **Finish** button.
9. While still on the PostgreSQL server, configure the firewall for port 5432.
- In the Windows Server *Start* menu, enter **Windows Firewall with Advanced Security** in the search box. The *Windows Firewall with Advanced Security* window displays.

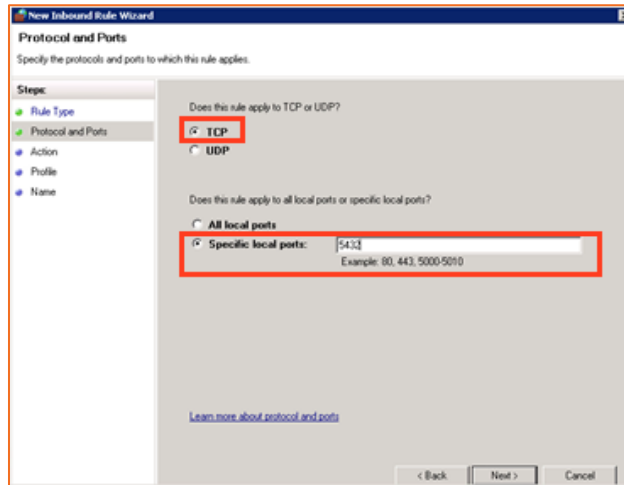


- b. Right-click the *Inbound Rules* navigation item and select **New Rule**. The *New Inbound Rule Wizard* displays.
- c. In the *New Inbound Rule Wizard*, select the **Port** radio button, and then click the **Next** button. The wizard advances to the *Protocol and Ports* screen.

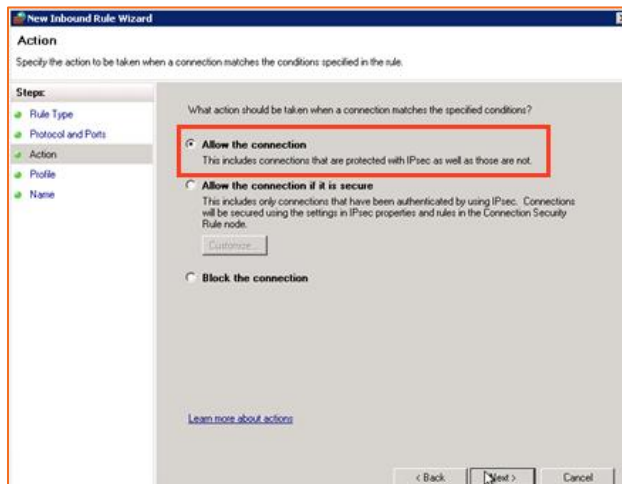


- d. In the *Protocol and Ports* screen, configure the following options:
 - i. Select the **TCP** radio button.
 - ii. Select the **Specific local ports** radio button.
 - iii. Enter port **5432**.

- iv. Click the **Next** button to continue to the *Action* screen.

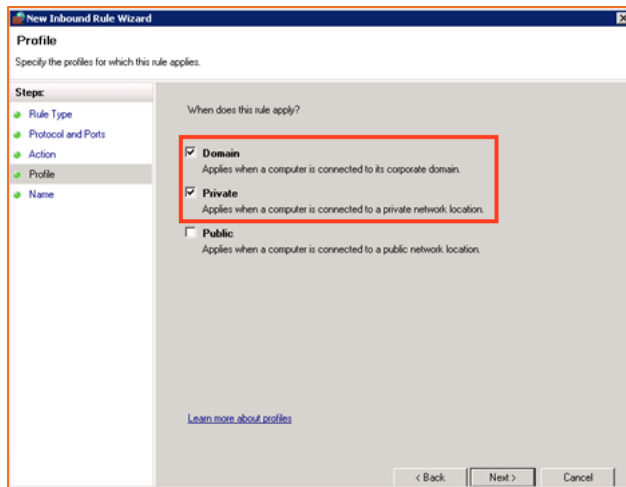


- e. In the *Action* screen, click the **Allow the connection** radio button, and click the **Next** button.



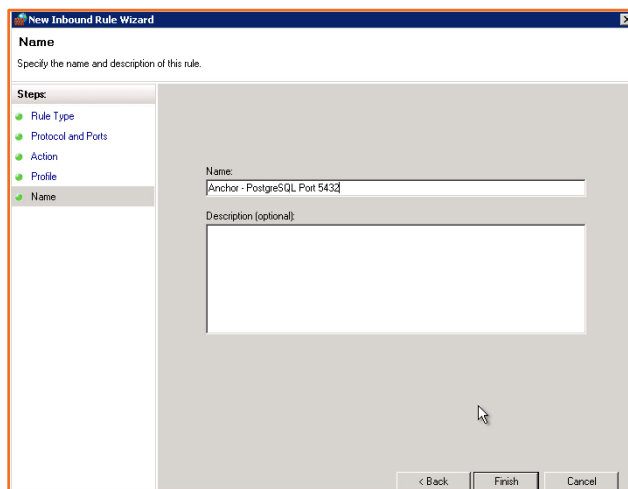
The *Profile* screen displays.

- f. In the *Profile* screen, ensure that *Domain* and *Private* are selected, and *Public* is unselected. Then, click the **Next** button to continue.



The *Name* screen displays.

- g. In the *Name* screen, enter an appropriate **name**, and optionally, a **description** of the new rule.

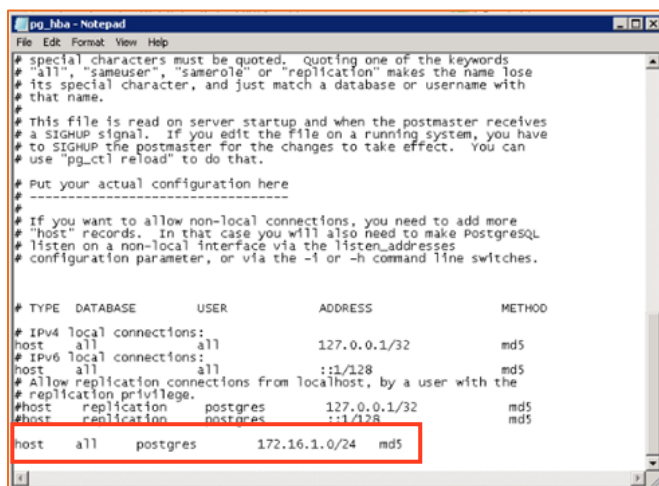


- h. Click the **Finish** button.

10. While still on the PostgreSQL server, configure a PostgreSQL configuration file, so that the PostgreSQL server knows to accept connections from another machine.
- Navigate to the *PostgreSQL* installation directory, which will be located at the default installation location (for example, `C:\PostgreSQL9.1`).
 - In the *PostgreSQL* directory, click the **data** folder, and then open the **pg_hba.conf** file.

- c. In the *PostgreSQL* configuration file, scroll down to the bottom of the file and add the following configuration parameters that specify the machines, databases, and users that can connect to PostgreSQL, including:
- Type of connection (for example, a *host* connection);
 - Type of database (for example, *all* databases);
 - The user;
 - The IP address of the current server—in standard dotted decimal notation—and an CIDR Mask Link, which indicates the number of significant bits that must make up the client IP address (for example, *172.16.1.0/24*); and
 - The hash algorithm (for example, *md5*).

For more information on PostgreSQL client authentication and the `pg_hba.conf` file, please review the [PostgreSQL manual](#).



```

pg_hba - Notepad
File Edit Format View Help
# special characters must be quoted, quoting one of the keywords
# "all", "sameuser", "samerole" or "replication" makes the name lose
# its special character, and just match a database or username with
# that name.
#
# This file is read on server startup and when the postmaster receives
# a SIGHUP signal. If you edit the file on a running system, you have
# to SIGHUP the postmaster for the changes to take effect. You can
# use "pg_ctl reload" to do that.
#
# Put your actual configuration here
#
# If you want to allow non-local connections, you need to add more
# "host" records. In that case you will also need to make PostgreSQL
# listen on a non-local interface via the listen_addresses
# configuration parameter, or via the -l or -h command line switches.
#
# TYPE DATABASE USER ADDRESS METHOD
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5
host all postgres 172.16.1.0/24 md5

```

- Save and close the *PostgreSQL* configuration file.
- Restart the *PostgreSQL* service.

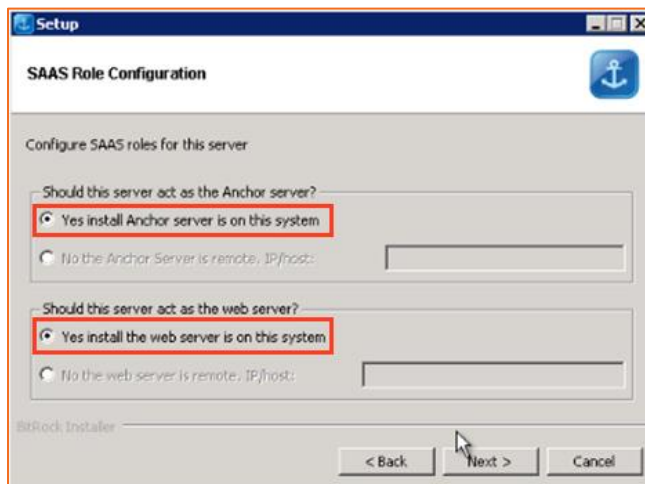


Note: During installation, our installer will automatically attempt to set these parameters properly, using primary IP address of the machine, and will create Portal database and single Root database on this server. However, if you have multiple networking interfaces you might still need to adjust these manually.

Step 2: Install Anchor and Apache on Machine 2

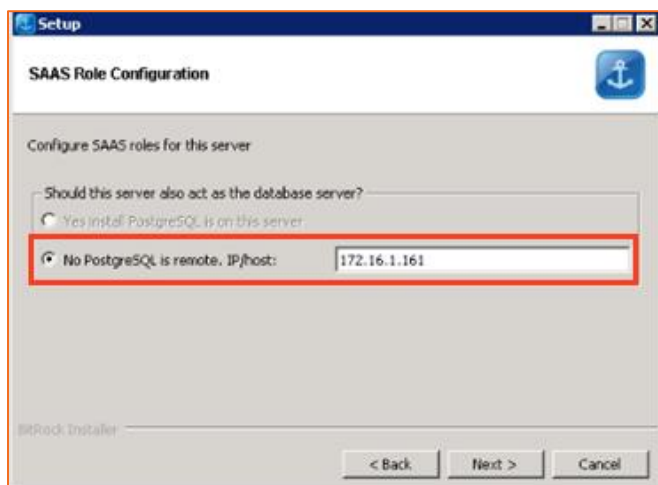
After Anchor and PostgreSQL are installed on machine 1, you can configure machine 2. For this example, we will install Anchor and Apache servers on machine 2.

1. Log into machine 2 (the Anchor and Apache server) as a local admin.
2. In the server, download the latest Anchor Server installation file.
3. Open an elevated Command Prompt window. The Command Prompt window displays.
4. In the Command Prompt window, run the Anchor Server executable with the `--saas 1` argument, which will allow you to select specific components to install during the installation process. The *Anchor Setup Wizard* will launch.
5. In the *Anchor Setup Wizard*, click the series of **Next** buttons to proceed through the setup process, making sure to pause at the *SAAS Role Configuration* screen.
6. In the *SAAS Role Configuration* screen, select the **Yes install Anchor server is on this system** radio button, the **Yes install the web server is on this system** radio button, and then click the **Next** button to continue.



7. In the next *SaaS Role Configuration* screen, select the **No PostgreSQL is remote** radio button, enter the **IP address** or **hostname** of the Anchor and PostgreSQL server (machine

1), and then click the **Next** button to continue.



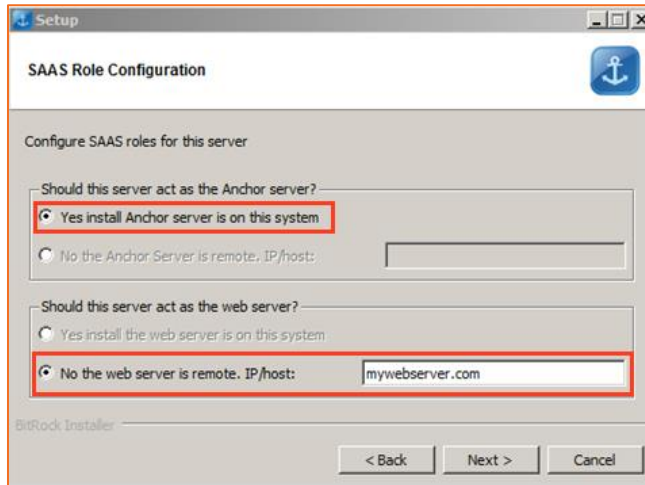
8. During the installation process, a *Question* dialog box will display, asking you if you would like to use the existing database the installer found on the system. Select the **Yes** button to confirm and continue.
9. Click the **Finish** button when the installation process is complete.

Step 3: Install Anchor on Machine 3

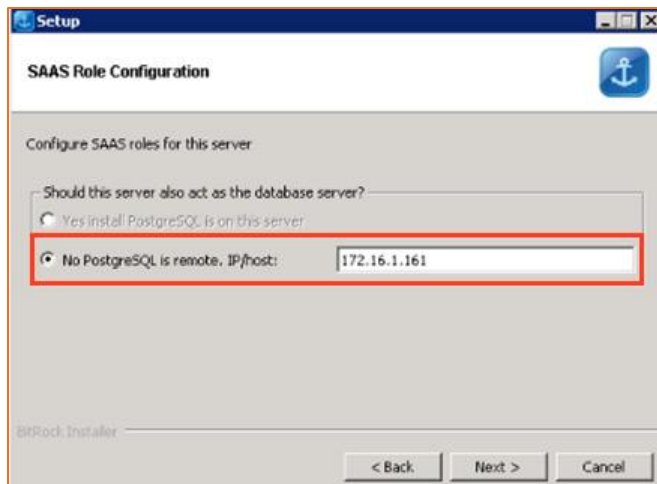
After you install Anchor and Apache on machine 2, you can install the third instance of Anchor on machine 3.

1. Log into machine 3 (the Anchor server) as a local admin.
2. In the server, download the latest Anchor installation file.
3. Open an elevated Command Prompt window. The Command Prompt window displays.
4. In the Command Prompt window, run the Anchor Server executable with the `--saas 1` argument, which will allow you to select specific components to install during the installation process. The *Anchor Setup Wizard* will launch.
5. In the *Anchor Setup Wizard*, click the series of **Next** buttons to proceed through the setup process, making sure to pause at the *SAAS Role Configuration* screen.
6. In the *SAAS Role Configuration* screen, select the following options:
 - a. Select the Yes install Anchor server on this system radio button.
 - b. Select the No the web server is remote radio button.
 - c. Enter the web server **IP address** or **hostname**.

- d. Click the **Next** button when you are finished.



7. In the next *SAAS Role Configuration* screen, select the **No PostgreSQL is remote** radio button, enter the **IP address** or **hostname** of the Anchor and PostgreSQL server (machine 1), and then click the **Next** button to continue.



8. During the installation process, a *Question* dialog box will display, asking you if you would like to use the existing database the installer found on the system. Select the **Yes** button to confirm and continue.
9. Click the **Finish** button when the installation process is complete.

Option 1: Configuring a Locally Mounted Storage Drive



Note: These steps are only applicable if you plan to use a locally mounted drive. In most instances—especially for advanced environments—it is recommended that you plan to use a NAS or SAN storage device. For more information, please reference the [Option 2: Configuring a NAS or SAN Storage Device](#) section of this guide.

After the server services are installed, you need to create a location for the root store and the file store. You must then map these stores to each of the machines under the system context. For the purposes of this document, we will set up file stores on machine 1, and map these stores to machine 2 and machine 3.

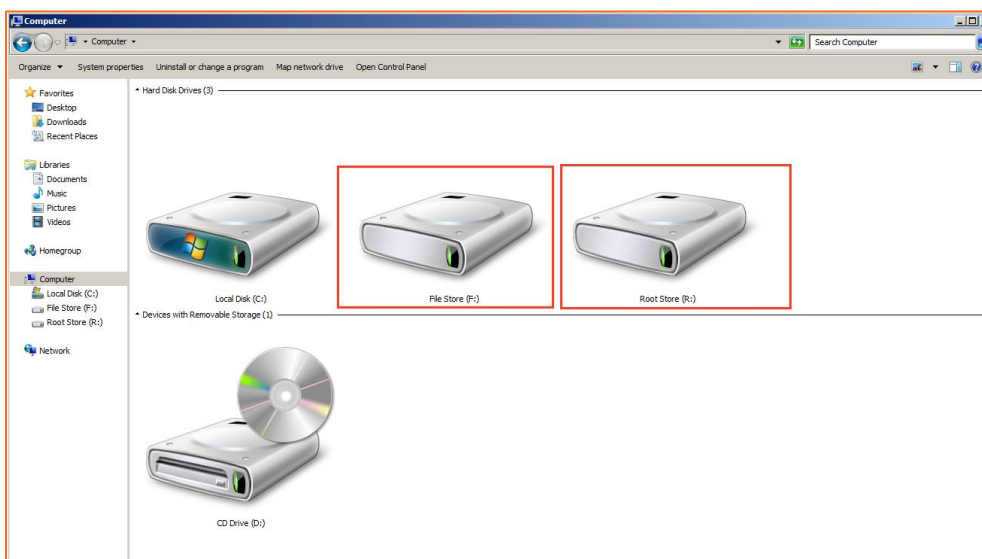
Configuration Instructions

Step 1: Map Drives



Note: When mapping drives, the paths must be the same (same drive letter) on each of the servers. Additionally, these drives need to be mapped under the system context.

1. On machine 1 (the Anchor and PostgreSQL server), create a root store location and a file store location.



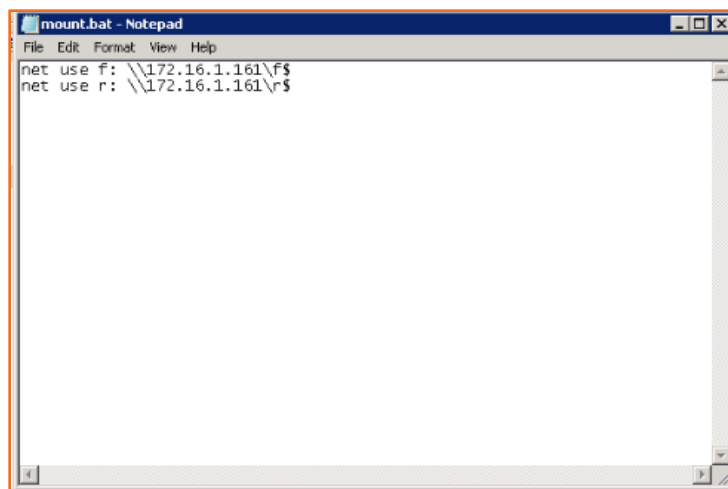
2. On machine 2 (the Anchor and Apache server), map a drive under the system context to the root and file stores on machine 1.



Note: The following script will *not* function when the server restarts or is unavailable.

- a. Navigate to the Anchor server installation drive (for example, `C:\Anchor Server`), and create a new batch file titled, `mount.bat`.
- b. Open `mount.bat` and use the **net use** command to map the drives locally to machine 1 as follows:


```
net use <drive letter>: \\<ip_of_storage_server>\<roots_mount_name>
net use <drive letter>: \\<ip_of_storage_server>\<files_mount_name>
```

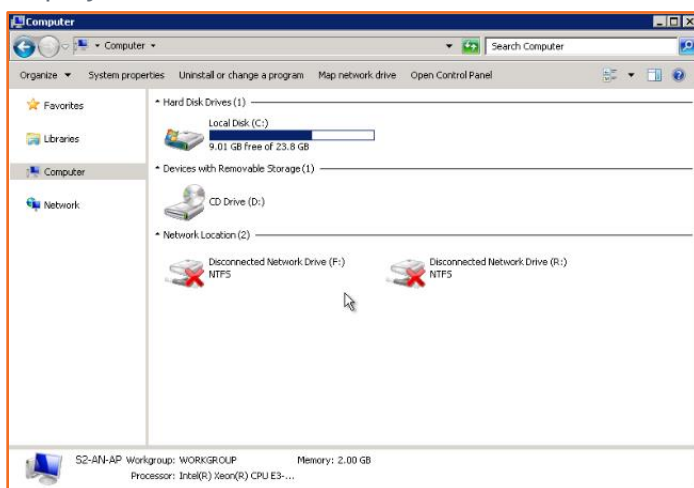


```
mount.bat - Notepad
File Edit Format View Help
net use f: \\172.16.1.161\F$
net use r: \\172.16.1.161\R$
```

- c. Save and close the file.
 - d. While still on machine 2, in the Windows Server *Start* menu, enter **gpedit.msc** in the *search* box. The *Local Group Policy Editor* window displays.
 - e. In the *Local Group Policy Editor* window, click **Computer Configuration**, then click **Windows Settings**, then click **Scripts**, and then select **Startup**. The *Startup Properties* dialog box displays.
 - f. In the *Startup Properties* dialog box, click the **Add** button. The *Add a New Script* dialog box displays.
 - g. In the *Add a New Script* dialog box, select the *mount.bat* file that was just created.
 - h. Click the **OK** button and close all of the dialog boxes and windows.
3. On machine 3 (the Anchor server), map a drive under the system context to the root and file stores on machine 1.
 - a. Navigate to the Anchor server installation drive (for example, *C:\Anchor Server*), and create a new batch file titled, *mount.bat*.
 - b. Open *mount.bat* and use the **net use** command to map the drives locally to machine 1 as follows:

```
net use <drive letter>: \\<ip_of_storage_server>\<roots_mount_name>
net use <drive letter>: \\<ip_of_storage_server>\<files_mount_name>
```
 - c. Save and close the file.
 - d. While still on machine 3, in the Windows Server *Start* menu, enter **gpedit.msc** in the *search* box. The *Local Group Policy Editor* window displays.

- e. In the *Local Group Policy Editor* window, click **Computer Configuration**, then click **Windows Settings**, then click **Scripts**, and then select **Startup**. The *Startup Properties* dialog box displays.
 - f. In the *Startup Properties* dialog box, click the **Add** button. The *Add a New Script* dialog box displays.
 - g. In the *Add a New Script* dialog box, select the *mount.bat* file that you just created.
 - h. Click the **OK** button and close all of the dialog boxes and windows.
4. Restart machine 2 and machine 3.
 5. When you log in to machine 2 and machine 3, you will see the mapped drives displayed as follows:



Step 2: Register Anchor

Now that your server services are installed, and storage locations are configured, you can set up the Anchor system. For full instructions on how to set up the Anchor system, please reference the [Anchor System Setup Knowledgebase](#) article.

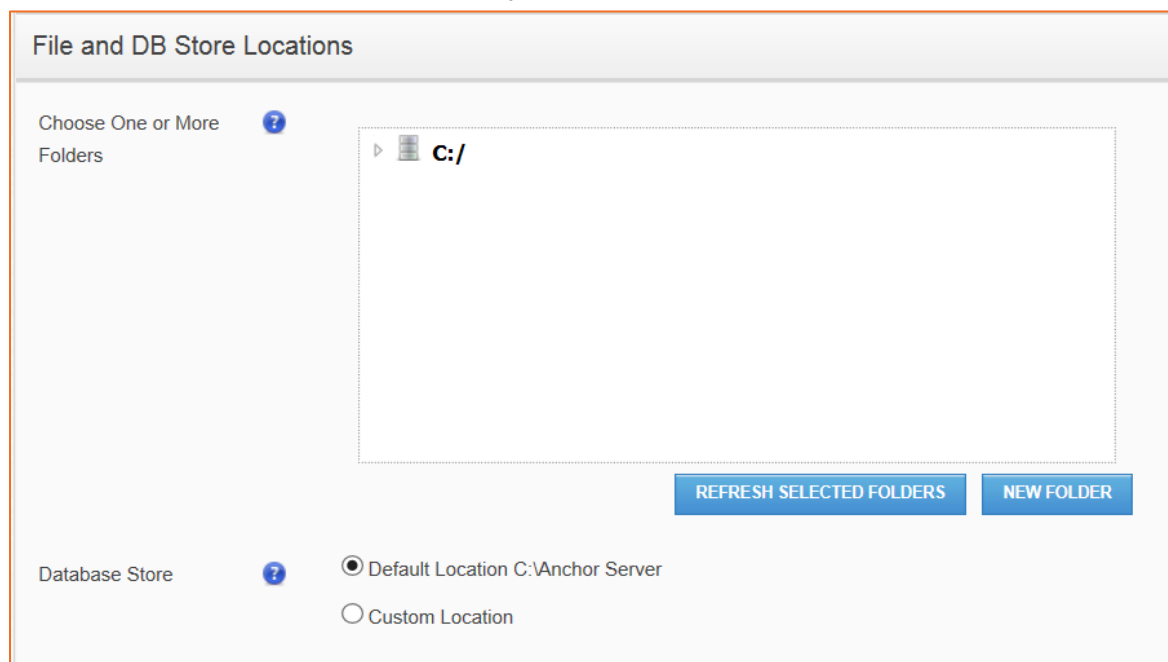
1. On the Anchor and Apache server machine (for our example, machine 2), click the **Anchor Server Management** desktop shortcut. The *administrative web portal* launches in your browser.
2. In the *License* tab and *System Admin* tab, enter the required information.
3. In the *System Settings* tab, you must configure hostname settings for the load balancer, as well as the file and DB store locations.
 - a. In the *Hostname* field, enter the **hostname** of the load balancer.

- b. Use the *Dual Hostname* checkbox and *App Server Hostname* field to configure a different hostname for the app server and web server. You can configure these settings after you complete the initial setup process. For more information, please reference [Configuring Dual Hostname Settings](#) section of this guide.

License	System Admin	System Settings	Encryption	Email Server
Hostname or IP				
<p> Note: The IP address or hostname (recommended) you set below will be embedded in the agent installation. Agents will not be able to connect to the server if the hostname does not properly resolve. If you plan on using unique hostnames for each customer, you will be able to specify those after system set-up.</p>				
Hostname		<input type="text"/>	*	
Dual Hostnames		<input type="checkbox"/>	Use different hostnames for the app server and web server.	
App Server Hostname		<input type="text"/>		

- c. In the *File and DB Store Locations* section, navigate to the **file store location** created in the steps above.

- d. In the *Database Store* field, select **Custom Location**, and then navigate to the **root store location** created in the steps above.

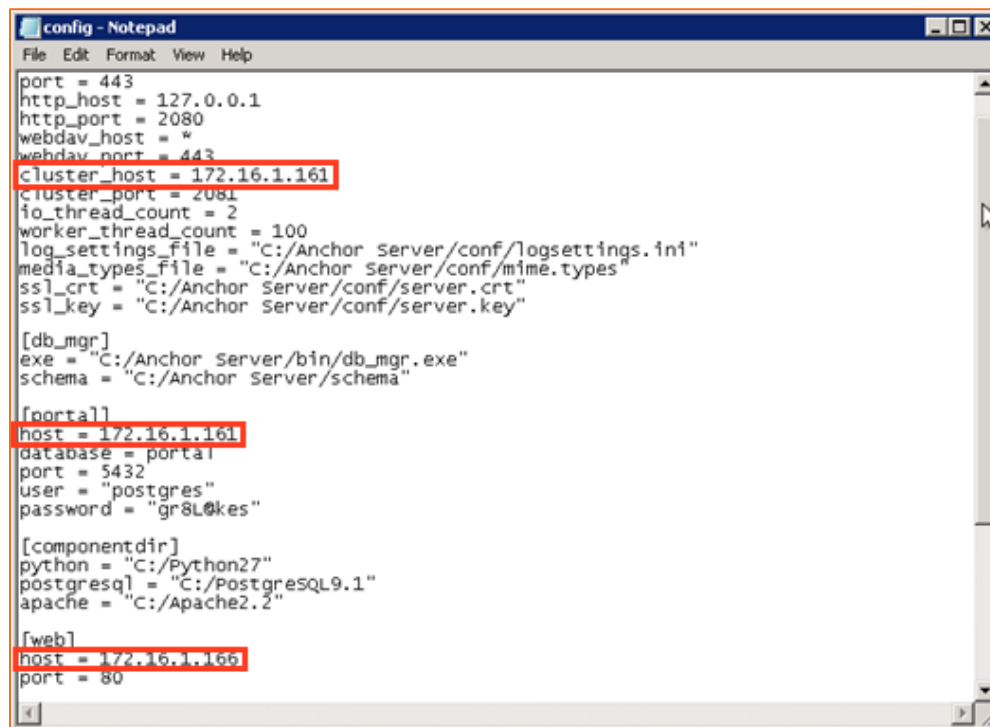


Step 3: Modify Anchor Configuration Files

After the Anchor system has been set up, you must modify the Anchor configuration file on each machine, and then restart each Anchor Server service.

1. On machine 1 (the Anchor and PostgreSQL server), modify the Anchor configuration file.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.
 - c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host—set this parameter to the IP address of the current machine (in this case, machine 1, or the Anchor and PostgreSQL server).
 - ii. Portal host—set this parameter to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host—this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).

- d. Save and close the *config.ini* file.



```

config - Notepad
File Edit Format View Help
port = 443
http_host = 127.0.0.1
http_port = 2080
webdav_host = *
webdav_port = 443
cluster_host = 172.16.1.161
cluster_port = 2081
io_thread_count = 2
worker_thread_count = 100
log_settings_file = "C:/Anchor Server/conf/logsettings.ini"
media_types_file = "C:/Anchor Server/conf/mime.types"
ssl_cert = "C:/Anchor Server/conf/server.crt"
ssl_key = "C:/Anchor Server/conf/server.key"

[db_mgr]
exe = "C:/Anchor Server/bin/db_mgr.exe"
schema = "C:/Anchor Server/schema"

[portal]
host = 172.16.1.161
database = portal
port = 5432
user = "postgres"
password = "gr8L0kes"

[componentdir]
python = "C:/Python27"
postgresq = "C:/PostgreSQL9.1"
apache = "C:/Apache2.2"

[web]
host = 172.16.1.166
port = 80

```

2. On machine 2 (the Anchor and Apache server), repeat the same process.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.
 - c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host – this parameter will be set as the IP address of the current machine (in this case, machine 2, or the Anchor and Apache server).
 - ii. Portal host – this parameter will be set to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host – this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).
 - d. Save and close the *config.ini* file.
3. On machine 3 (the Anchor server), repeat the same process.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.

- c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host – this parameter will be set as the IP address of the current machine (in this case, machine 3, or the Anchor server).
 - ii. Portal host – this parameter will be set to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host – this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).
 - d. Save and close the *config.ini* file.
4. Restart the Anchor server service on each machine.

Option 2: Configuring a NAS, SAN, or Other Storage Device

In instances where storage is set up on a NAS, SAN, or other storage device, you will need to enable the use of a UNC path so that each server can access the root store and the file store.

For the purposes of this document, we will describe the process of enabling and using a UNC path within a three server, high-availability environment (using Windows Server 2008 or 2012), as outlined below:

- Machine 1: Anchor server and PostgreSQL
- Machine 2: Anchor server and Apache server
- Machine 3 (or more): Anchor server



Note: This process requires the creation and use of a local admin account on each server; therefore, each server must be a non-domain controller (DC) server.

Configuration Instructions

Step 1: Create Local Admin Accounts on Each Machine

As a first step, create a local administrator account on each machine.



Note: Each local admin account *must* have the same username and password. For instructions on how to create a local admin account, refer to the Microsoft Technet articles, [Create a Local User Account](#) and [Add a Member to a Local Group](#).

Step 2: Register Anchor

1. On the Anchor and Apache server machine (for our example, machine 2), click the **Anchor Server Management** desktop shortcut. The *administrative web portal* launches in your browser.
2. In the *License* tab and *System Admin* tab, enter the required information.

3. In the *System Settings* tab, you must configure hostname settings for the load balancer, as well as the file and DB store locations.
 - a. In the *Hostname* field, enter the **hostname** of the load balancer.
 - b. Use the *Dual Hostname* checkbox and *App Server Hostname* field to configure a different hostname for the app server and web server. You can configure these settings after you complete the initial setup process. For more information, please reference [Configuring Dual Hostname Settings](#) section of this guide.

The screenshot shows the 'System Settings' tab in the Axcient interface. The 'Hostname or IP' section is highlighted with a red border. It contains a yellow warning box with a triangle icon and the following text: 'Note: The IP address or hostname (recommended) you set below will be embedded in the agent installation. Agents will not be able to connect to the server if the hostname does not properly resolve. If you plan on using unique hostnames for each customer, you will be able to specify those after system set-up.' Below the warning box, there are three input fields: 'Hostname' with a blue question mark icon and an asterisk, 'Dual Hostnames' with a blue question mark icon and a checkbox labeled 'Use different hostnames for the app server and web server.', and 'App Server Hostname' with a greyed-out input field.

- c. In the *File and DB Store Locations* section, create a temporary folder for configuration purposes (for example, c:\Anchor Server\file_store_1). You will later move the location of this folder.
- d. In the *Database Store* field, select the **Default Location** radio button to create a temporary folder for configuration purposes (for example, c:\Anchor

Server\store_1). You will later move the location of this folder.

Step 3: Copy Metadata

When Anchor installation and registration is complete, and you are able to access the web portal, copy metadata from the temporary folders created above to the new location of your choice (for example, a location within a NAS or SAN device).

1. Copy all contents in the temporary file store (for example `c:\Anchor Server\file_store_1`) to the permanent file store location.
2. Copy all contents in the temporary root store (for example, `c:\Anchor Server\store_1`) to the permanent root store location.

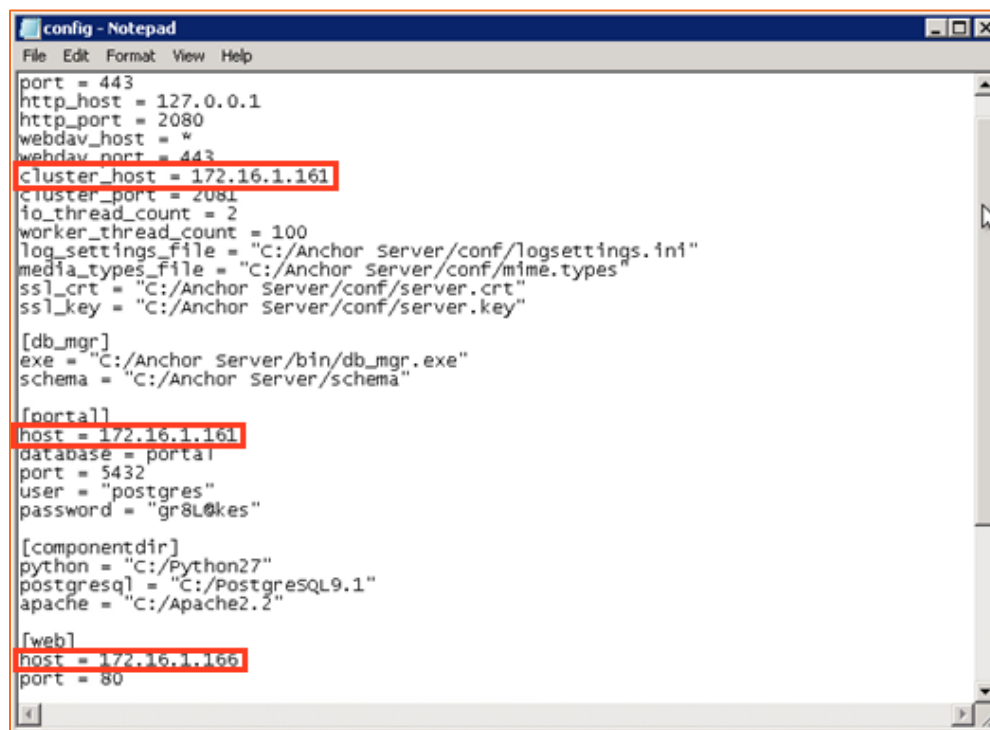
Step 4: Change Services to Run as Local Admin

1. For each Anchor server service, change the user under which the service is running to the local admin created above.
2. Restart all services.

Step 5: Modify Anchor Configuration Files

After the Anchor system has been set up, you must modify the Anchor configuration file on each machine, and then restart each Anchor Server service.

1. On machine 1 (the Anchor and PostgreSQL server), modify the Anchor configuration file.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.
 - c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host—set this parameter to the IP address of the current machine (in this case, machine 1, or the Anchor and PostgreSQL server).
 - ii. Portal host—set this parameter to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host—this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).
 - d. Save and close the *config.ini* file.



```

config - Notepad
File Edit Format View Help
port = 443
http_host = 127.0.0.1
http_port = 2080
webdav_host = *
webdav_port = 443
cluster_host = 172.16.1.161
cluster_port = 2081
io_thread_count = 2
worker_thread_count = 100
log_settings_file = "C:/Anchor Server/conf/logsettings.ini"
media_types_file = "C:/Anchor Server/conf/mime.types"
ssl_cert = "C:/Anchor Server/conf/server.crt"
ssl_key = "C:/Anchor Server/conf/server.key"

[db_mgr]
exe = "C:/Anchor Server/bin/db_mgr.exe"
schema = "C:/Anchor Server/schema"

[portal]
host = 172.16.1.161
database = portal
port = 5432
user = "postgres"
password = "gr8L0kes"

[componentdir]
python = "C:/Python27"
postgresql = "C:/PostgreSQL9.1"
apache = "C:/Apache2.2"

[web]
host = 172.16.1.161
port = 80
  
```

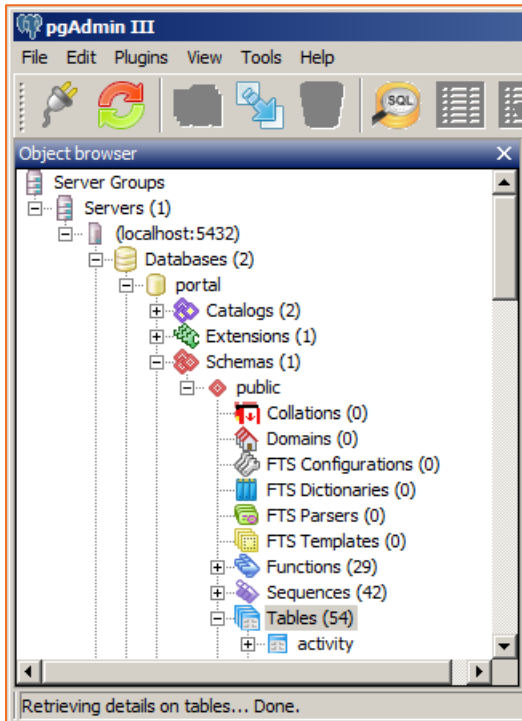
2. On machine 2 (the Anchor and Apache server), repeat the same process.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.

- c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host – this parameter will be set as the IP address of the current machine (in this case, machine 2, or the Anchor and Apache server).
 - ii. Portal host – this parameter will be set to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host – this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).
- d. Save and close the *config.ini* file.
3. On machine 3 (the Anchor server), repeat the same process.
 - a. Open the *Anchor Server* installation directory (for example, *C:\Anchor Server*).
 - b. In the *Anchor Server* directory, click the **conf** folder, and then open the **config.ini** file. The *config.ini* file displays.
 - c. Change the *cluster host*, the *portal host*, and the *web host* parameters in the *config.ini* file.
 - i. Cluster host – this parameter will be set as the IP address of the current machine (in this case, machine 3, or the Anchor server).
 - ii. Portal host – this parameter will be set to the explicit IP address of the Anchor and PostgreSQL server (machine 1).
 - iii. Web host – this parameter will be set to the explicit IP address of the Anchor and Apache server (machine 2).
 - d. Save and close the *config.ini* file.
4. Restart the Anchor server service on each machine.

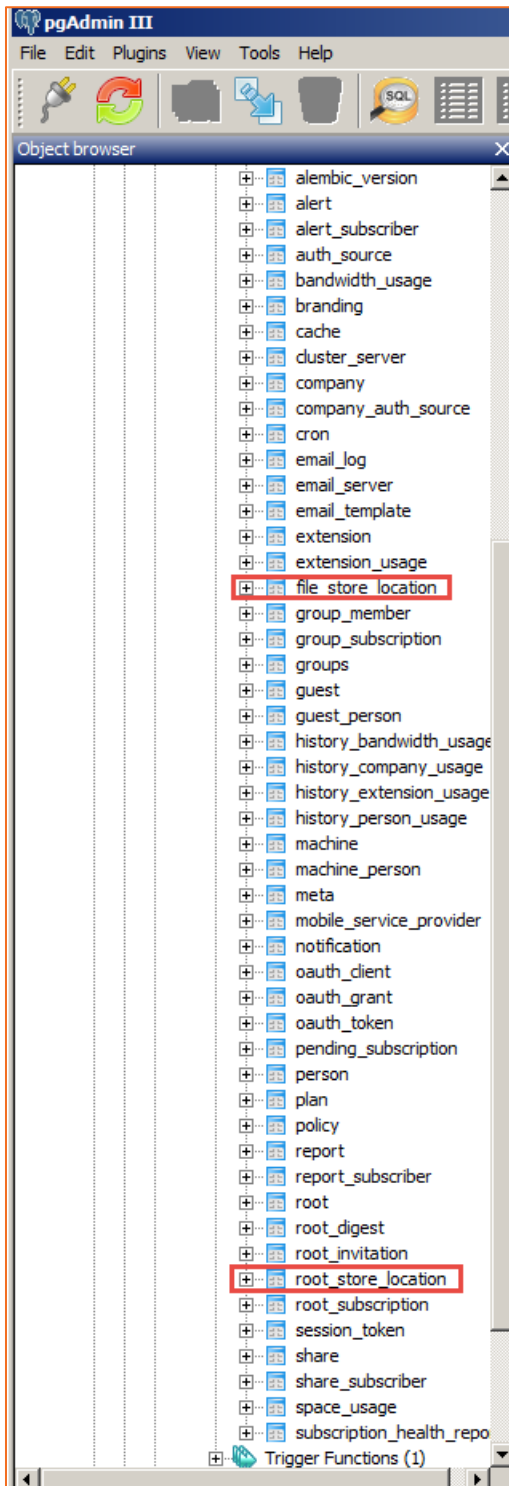
Step 6: Update PostgreSQL with the UNC Path

1. On machine 1 (Anchor server and PostgreSQL server) navigate to *C:\PostgreSQL9.1\bin*, and launch **pgAdmin3**. The application will open, prompting you to log in (password: *gr8L@kes*).
2. In the Object Browser, click to expand **Databases**, then click to expand **Portal**, then click to expand **Schemas**, then click to expand **Public**, and finally, click to expand

Tables.

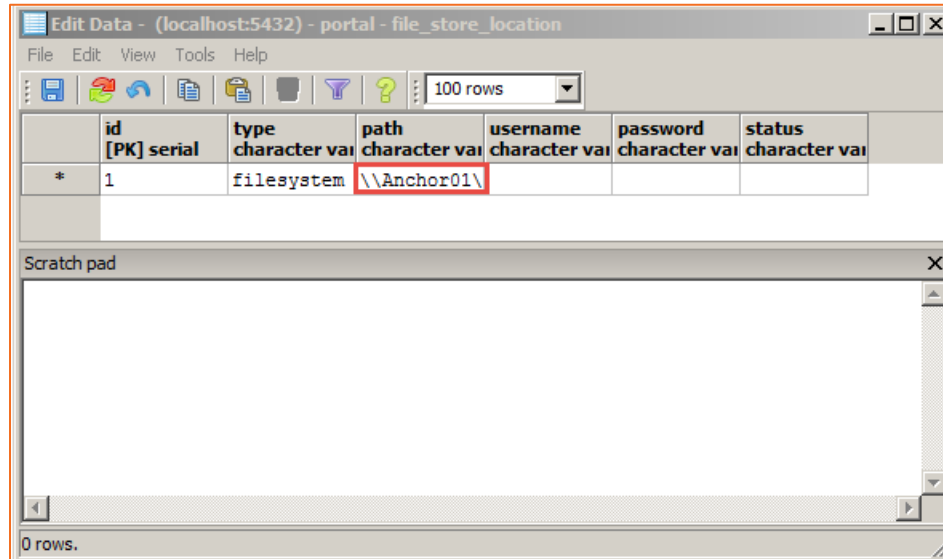


3. In the *Tables* section, locate the File and Root store tables.



4. In both the File Store and Root Store tables, update the path location.

- a. Right click the table, point to *View Data*, and select **View Top 100 rows**.
- b. In the *path* field, enter the **UNC path**.
- c. Save your changes.



5. Restart all services.

Configuring Backups

It is recommended that you back up the root store, the file store, and PostgreSQL.

For backup of root store and file store data it's recommended to use Axcient BCDR solutions such as x360Recover:

<https://axcient.com/products/x360recover/>

For information about configuring backups of PostgreSQL databases, refer to PostgreSQL Documentation:

<https://www.postgresql.org/docs/14/backup.html>

Reviewing Mappings for the Load Balancer or DNS Server

After storage is set up in your environment, you can review IP mappings for the load balancer or DNS server. Under the default settings, port 443 is reserved for the Anchor service, and ports 80 and 510 are reserved for the Apache service.

Default Port Settings for an Anchor Server:

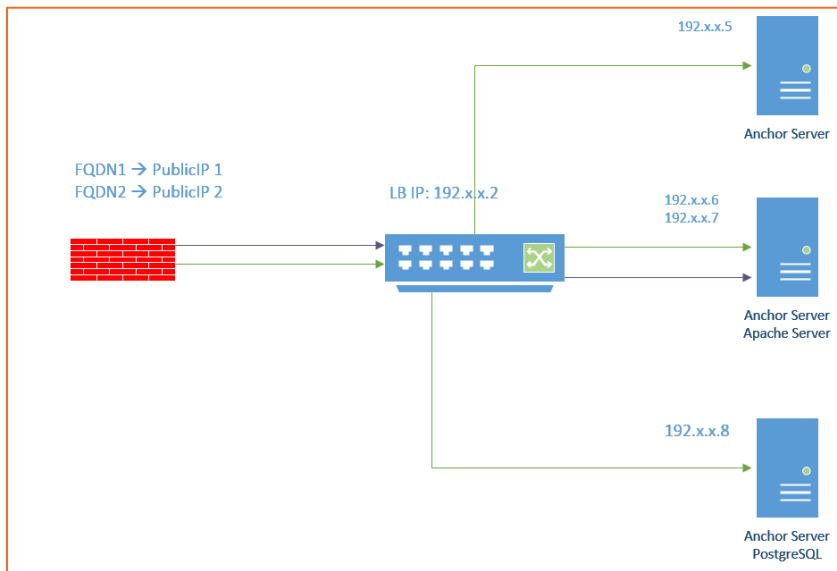
External_IP_1 FQDN1:443 | Load_Balancer_IP → Internal_IP_1:443

Default Port Settings for an Apache Server:

External_IP_1 FQDN1:510 | Internal_IP_1:510

External_IP_1 FQDN1:80 | Internal_IP_1:80

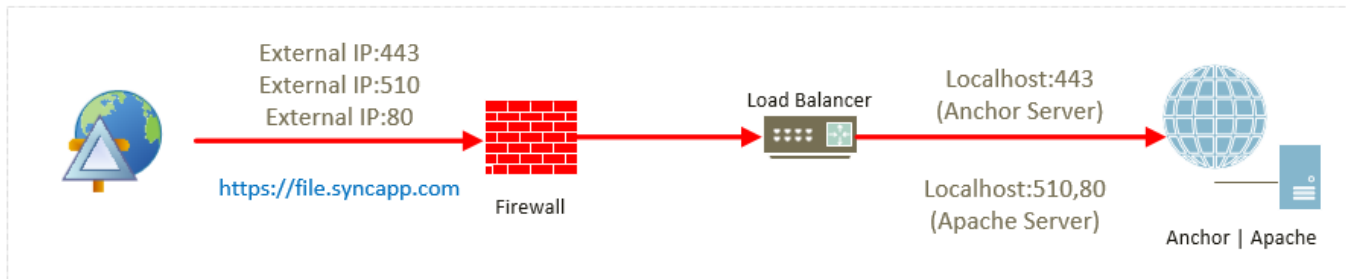
For security purposes, it is recommended that you remove port 510 and instead utilize port 410 through the use of dual hostname settings. For more information, please reference the [Configuring Dual Hostname Settings](#) section of this guide.



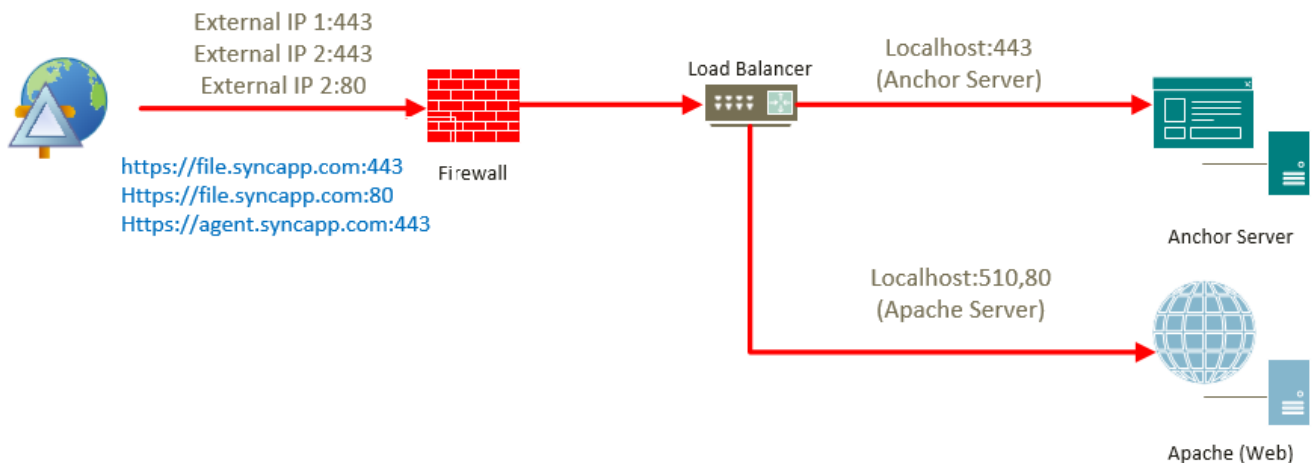
Configuring Dual Hostname Settings

By default, the Anchor and Apache services are installed on one server with one shared domain (for example, <https://file.syncapp.com>) and run on the localhost. Port 443 can only be used by one service. Under the default settings, port 443 is reserved for the Anchor service, and ports 80 and 510 are reserved for the Apache service.

Default Port Settings for an Anchor Server:



As an alternative to this configuration, Anchor and Apache can be configured to use two separate domains (or use a domain and a subdomain, such as <http://SyncApp.com> and <http://web.SyncApp.com>). A dual Host Cert and two Public IP's is required. Under this configuration, both the Anchor service and the Apache service can allow external connections on 443, which improves access.



In this setup Port forwarding will be used to NAT traffic from <https://file.syncapp.com:443> to Apache Service Server IP 10.X.X.12:510. Apache will continue listening on port 510 internally.

If you would like to completely remove all traces of port 510 then either an additional internal IP will be needed to allow for the Anchor and Apache services to both listen on port 443 or Apache can be run on a VM that is not running anchor.

Configuration Instructions

Step 1: Configure a Second Domain or Subdomain



Note: If you have already deployed agents, you will need to reserve your original domain for the Anchor service and assign the new domain or subdomain to Apache; this means that you and your users will no longer be able to access the web portal through the existing domain. Agents are hard-coded and will not recognize a new domain without a full uninstall and reinstall of the agents.

Please contact Support if you are deploying this into a production environment.

As a first step, you will need to configure a second domain, or create a new subdomain, through your DNS hosting provider's web site. For specific instructions, please contact your DNS hosting provider.

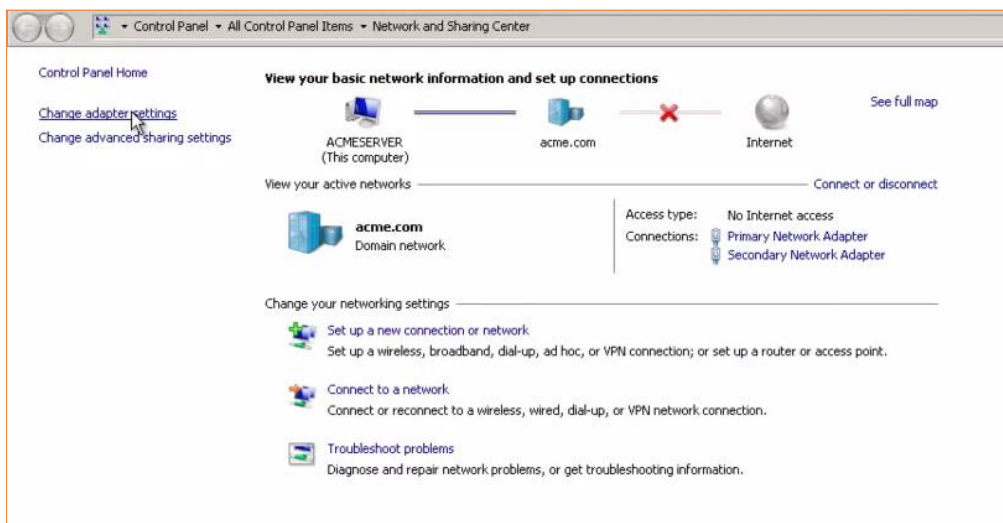
(Optional) Step 1a: Configure an Additional IP Address

This step is only necessary if you would like to completely remove all traces of port 510. After you configure a second domain or a subdomain, you can configure a new IP address. You do not need a new NIC.

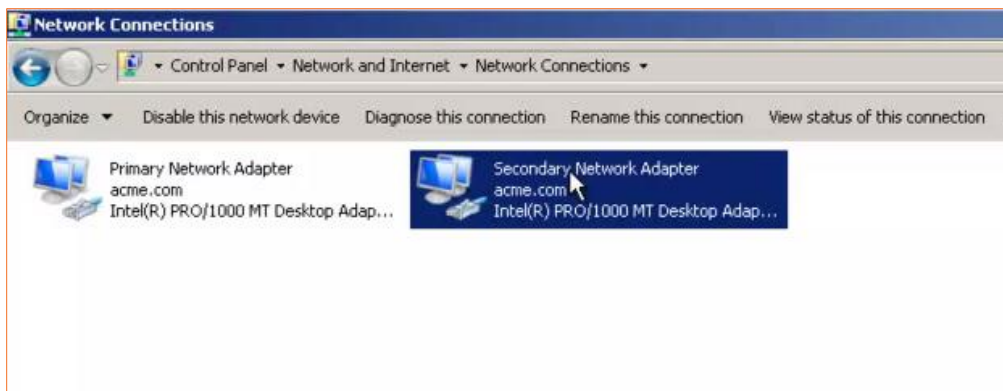


Note: These steps will help you configure an IP address at the OS-level. Alternatively, you can utilize a different configuration method (for example, network level configuration).

1. In the *Start* menu, point to *Control Panel*, and select **Network and Sharing Center**. The *Network and Sharing Center* window displays.

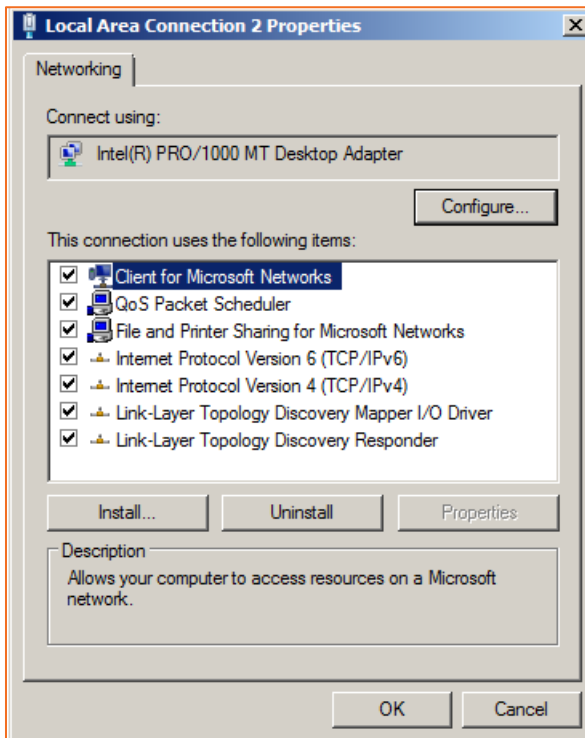


2. In the *Network and Sharing* window, click **Change Adapter Settings**. The *Network Connections* window displays.



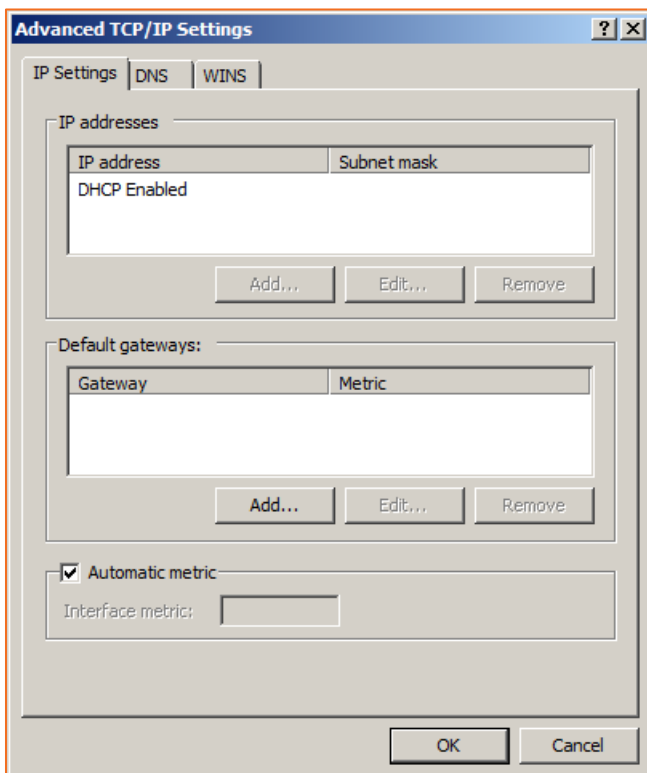
3. In the *Network Connections* window, right-click the **network adapter** to which the IP address will be added, and select **Properties**. The *Local Area Connection Properties*

dialog box displays.

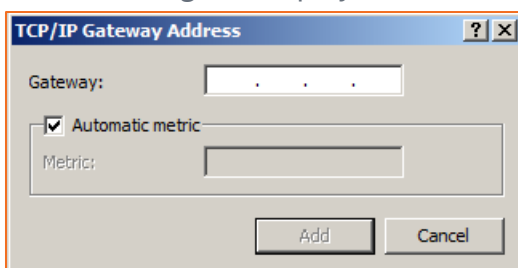


4. In the *Local Area Connection Properties* dialog box, click to highlight **Internet Protocol Version 4 (TCP/IPv4)**. With *Internet Protocol Version 4 (TCP/IPv4)* highlighted, click the **Properties** button. The *Internet Protocol Version 4 (TCP/IPv4) Properties* dialog box displays.

- In the Internet Protocol Version 4 (TCP/IPv4) Properties dialog box, click the **Advanced** button. The Advanced TCP/IP Settings dialog box displays.



- In the *Advanced TCP/IP Settings* dialog box, click the **Add...** button. The *TCP/IP Address* dialog box displays.



- In the *TCP/IP Address* dialog box, enter the **IP address** and **Subnet mask**, and click the **Add** button. The *Advanced TCP/IP Settings* dialog box now shows the new *IP address* and *Subnet mask*.
- Click the **OK** or **Close** buttons to return to the *Network and Sharing Center* window.

(Optional) Step 1b: Stop the Apache and Anchor Services

After you configure a new domain and a new IP address, you will need to stop the Apache and Anchor services.



Note: Ensure that any affected customer knows of this temporary downtime before stopping the services.



Note: The Apache service should be stopped before the Anchor service.

1. On the Apache server, launch *services.msc*.
2. In the *Services* window, right-click the **Apache** service and select **Stop**. The Apache service is now stopped.
3. While still in the *Services* window, right-click the **Anchor** service and select **Stop**. The Anchor service is now stopped.

Step 2: Point the New Domain or Subdomain to the new IP Address

When the services have been stopped, you can then point your new domain to the new IP address configured in the steps above.

For specific instructions, please reference guidelines provided by your web hosting provider. For example, [GoDaddy Support for Updating your Domain Name IP Address](#).

Step 3: Configure the Port Address Translation (PAT) Settings

The following PAT settings need to be configured:

file.syncapp.com:443 → External_IP1:443 → localhost:510 → Apache

file.syncapp.com:80 → External_IP1:80 → localhost:80 → Apache

agent.syncpp.com:443 → External_IP2:443 → localhost:443 → Anchor service

You can remove Port 510 completely, assuming Anchor Server and Apache are running on the same VM:

file.syncapp.com:443 → External_IP1:443 → Internal IP 1:443 → Apache

file.syncapp.com:80 → External_IP1:80 → localhost:80 → Apache (Apache does the redirect (httpd) and should always be set to listen on port 80 on the localhost)

Step 4: Configure the SSL Certificates

For information on configuring SSL certificates, please reference the [How Do I Configure a Single Domain or Wildcard SSL Certificate](#) Knowledgebase article.

Step 5: Update the Apache Configuration File

Next, the Apache configuration file will need to be updated with the new domain or subdomain.

1. In your Apache server, open the httpd.conf file, which is located at *[target drive]:\Apache24\conf*.
2. Copy and paste the file in the current location. This will create a backup of the current running Config file.
3. Compare the httpd.conf file to the http.conf file listed below. Make the necessary replacements, making sure to retain appropriate SSL certificate file paths.

```
Listen 80

LoadModule actions_module modules/mod_actions.so
LoadModule alias_module modules/mod_alias.so
LoadModule asis_module modules/mod_asis.so
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule authn_core_module modules/mod_authn_core.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authz_core_module modules/mod_authz_core.so
LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule cgi_module modules/mod_cgi.so
LoadModule env_module modules/mod_env.so
LoadModule include_module modules/mod_include.so
LoadModule isapi_module modules/mod_isapi.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule mime_module modules/mod_mime.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule ssl_module modules/mod_ssl.so
LoadModule socache_shmcb_module modules/mod_socache_shmcb.so
LoadModule wsgi_module modules/mod_wsgi.so
LoadModule rewrite_module modules/mod_rewrite.so

ErrorLog "logs/error.log"
```

```
LogLevel warn

<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\""
combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
    CustomLog "logs/access.log" common
</IfModule>

<IfModule mime_module>
    TypesConfig conf/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
</IfModule>

AllowEncodedSlashes On
TraceEnable Off
AcceptFilter http none
AcceptFilter https none
EnableSendfile Off
EnableMMAP Off

RewriteEngine On
RewriteCond %{HTTPS} off
RewriteCond %{REQUEST_URI} !^/updater
RewriteCond %{REQUEST_URI} !^/static/assets/
RewriteCond %{REQUEST_URI} !^/server/hostname
RewriteRule (.*) https://%{HTTP_HOST}443%{REQUEST_URI}

### SSL ###
Listen 510
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]

<VirtualHost _default_:510>
[REDACTED]
[REDACTED]
```

```

[REDACTED]
[REDACTED]
[REDACTED]
    SSLCertificateFile "conf/ssl/server.crt"
    SSLCertificateKeyFile "conf/ssl/server.key"
    SSLCertificateChainFile "conf/ssl/bundlecert.crt"
    AllowEncodedSlashes On
</VirtualHost>

<IfModule ssl_module>
    SSLRandomSeed startup builtin
    SSLRandomSeed connect builtin
</IfModule>
### END-SSL ###

WSGIPythonHome "C:/Anchor Server/penv"
WSGIScriptAlias / "C:/Anchor Server/web/anchor.wsgi" application-
group=%{GLOBAL}
WSGIImportScript "C:/Anchor Server/web/anchor.wsgi" application-group=%{GLOBAL}
WSGIPassAuthorization On

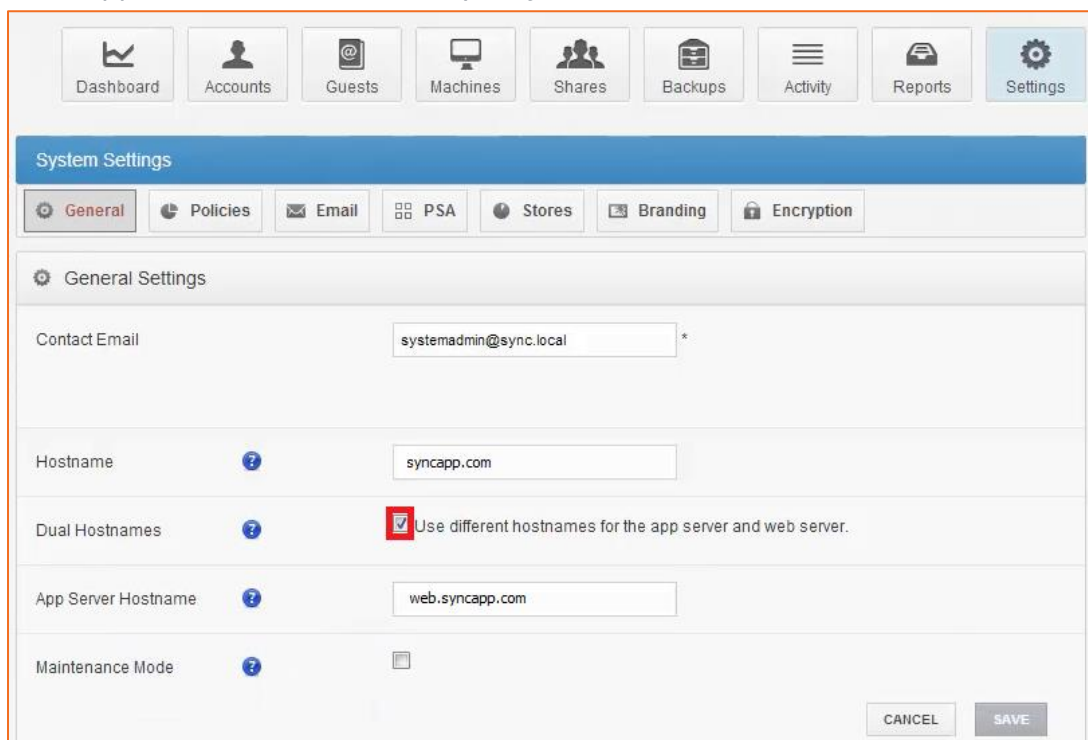
```

Save the file when you are finished and restart Apache server service. Apache should start with no exception. If there is an error, consult the event logs.

Step 6: Specify the App Server Hostname within the Web Portal

1. With both the Apache and Anchor services running, open the web portal in your browser.
2. From the web portal dashboard, click the **Settings** tab. The *Settings* page displays.
3. In the *Settings* page, click the **General** tab. The *General Settings* page displays.
4. In the *Dual Hostnames* field, click the **Dual Hostnames** checkbox.

5. In the *App Server Hostname* field, specify the new **app server hostname address**.



The screenshot shows the Axcient System Settings interface. At the top, there is a navigation bar with icons for Dashboard, Accounts, Guests, Machines, Shares, Backups, Activity, Reports, and Settings. Below this is a blue header for 'System Settings' with sub-tabs for General, Policies, Email, PSA, Stores, Branding, and Encryption. The 'General Settings' section is active, showing the following fields:

- Contact Email: systemadmin@sync.local *
- Hostname: syncapp.com
- Dual Hostnames: Use different hostnames for the app server and web server.
- App Server Hostname: web.syncapp.com
- Maintenance Mode:

At the bottom right, there are 'CANCEL' and 'SAVE' buttons.

6. Click the **Save** button when you are finished.
7. Restart the *Anchor* and *Apache* services.

After these steps are complete, email links will use the web domain name, and agents will connect to the app domain name.

Setting up PostgreSQL connection pooling

As it was said in the previous sections, with the Anchor cluster scaling up, the number of connections to the PostgreSQL servers will increase, as every Anchor Server requires 40-50 connections each. Every connection for the PostgreSQL server requires a shared memory, and with more than 150 connections, this can quickly become a problem. So, we strongly recommend setting up a connection pooler.

Connection pooler establishes a limited number of persistent connections to the PostgreSQL server and then works as an endpoint for client connections, processing incoming connections and proxy-executing the queries for clients without spawning the additional connections to the PostgreSQL server. This way, regardless of the number of client connections to the connection pooler, the number of connections on the PostgreSQL server remains roughly the same all the time which saves the memory and improves performance in the long run.

One such connection pooler which we can recommend to use is PGBouncer:

<https://www.pgouncer.org/>

It's a very lightweight connection pooler which can be installed on Linux or Windows and very easy to configure. It can be installed on a standalone server like we have it on our SaaS setup (note that this adds a bit of a latency to queries) or can be installed alongside the PostgreSQL, on the same server. In case you want to have it on standalone server, we recommend running Debian Linux on this one.

Installation of PGBouncer

Disregard the installation instructions of the PGBouncer site, these are intended for package maintainers. For an end user, you don't need to compile and build anything. For Linux, there are ready to use packages available for all the main Linux distributions such as Debian, RedHat, Ubuntu, Fedora and others.

So, for Debian and Debian-based distributions it's as simple as that:

```
sudo apt update
```

```
sudo apt install pgbouncer
```

For Windows, there are pre-build packages available from the official GitHub repository:

<https://github.com/pgbouncer/pgbouncer/releases>

In there, just pick up the Windows pre-built zip archive (named pgbouncer-<version>-windows-x86_64.zip) from the Assets section of latest stable release, then unpack on the target server and move on with configuration. It doesn't require any specific installation procedure; you will only need to unpack it where you see fit, and then add the PGBouncer binary to the Windows Firewall (or any other firewall software you might have on your server) rules to allow incoming connections to it from the local network.

Configuration of PGBouncer

PGBouncer comes with an example configuration file (pgbouncer.ini), which is documented directly in the file. Read this file together with the detailed configuration guide before moving on and make sure to understand the effect of all the main parameters:

<https://www.pgbouncer.org/config.html>

Then move on with setting everything up. You will need to adjust main configuration parameters and fill in the databases section of configuration file, adding there main "portal" database and any "root" databases you have created on your PostgreSQL server.

Here are some configuration tips for you:

1. **listen_addr = localhost** – change to `listen_addr = *` or `listen_addr = <IP_of_main_network_interface>` to be able to serve the connections from other servers.
2. **pool_mode = session** - in our SaaS, we're using `pool_mode = transaction` here but our number of servers is much higher. Check if performance will be good enough with session mode, if lacking, then switch to transaction mode later.
3. **default_pool_size** and **reserve_pool_size** - make sure the PostgreSQL server connection limits are adjusted accordingly. These parameters are per user/database pair, so it will allow these limits for every separate database listed in [databases] section. It can be overridden in the per-database configuration if needed, you can set individual limits for every DB.

Before switching from direct connection to PostgreSQL server to PGBouncer, check it out by manually connecting to the configured port from another node using any SQL client, run some queries, and make sure everything works as expected.

Configuration of Anchor servers to use connection pooler

From client connection perspective, connection pooler works as a normal PostgreSQL server, so no special configuration is required other than specifying the IP address and port of connection pooler in the configuration files.

For Anchor Servers, you need to edit configuration file:

C:\Anchor Server\conf\config.ini

And modify relevant parameters in section [portal]: set **host** to IP address or hostname of server running connection pooler, set **port** to connection pooler port you have assigned in the PGBouncer configuration.

For Apache Servers, you need to edit configuration file:

C:\Anchor Server\web\config.py

And modify relevant parameters: set **PORTAL_DB_HOST** to IP address or hostname of server running connection pooler, set **PORTAL_DB_PORT** to connection pooler port you have assigned in the PGBouncer configuration. **PORTAL_DB_PORT** parameter can be absent from configuration file by default, add it, if necessary, with an integer value.

For PostgreSQL Root databases, you need to modify values directly in PostgreSQL “**portal**” database, table “**root_db_location**”. Set values in the “**host**” and “**port**” columns of configured root databases, to IP address or hostname of server running connection pooler and port you have assigned in the PGBouncer configuration.

Scaling the number of Apache servers

During the operations of your HA cluster at some point it might be the case you will face the need to scale up the number of Apache servers in your environment to handle the increased load. Here is what needs to be done to implement that:

Deploying the new Apache servers

First, you will need install a couple more Apache servers. Install these running the Anchor Server executable with the `--saas 1` argument, which will allow you to select specific components to install during the installation process.

In the *SAAS Role Configuration* screen, select the **No the Anchor server is remote** radio button, the **Yes install the web server is on this system** radio button, configure the remote

Anchor server address and continue. On the next screen, select the **No PostgreSQL is remote** radio and configure the remote PostgreSQL server address.

When everything is installed, stop and disable Anchor Celery Service (there should be only ONE instance of running Anchor Celery Service per cluster) and stop and disable Redis (there also should be only ONE instance of running Redis per cluster) using Windows Services management (to disable the service, right-click on it, select Properties -> General, set Startup type - Disabled, then press Stop button, then OK). Also stop the Apache service and keep it stopped until all the configuration below is done.

More info on the configuring Anchor Celery Service and Redis is in the upcoming sections of this manual.

Setting up common shared static assets storage and shared services

First, you will need a **common assets storage** which will be used by all the Apache servers simultaneously. This storage will keep all the static assets for the web servers, including branding information for individual organizations registered in your system. It's crucial for all Apache servers to use the same shared storage and have it in a writeable state so that when some customer will register an organization using session on one of the web servers, assets for this organization will be created in a common storage and immediately accessible for all the Apache servers in your system.

To create such a storage, you can use following approach:

Step 1: Create a domain or regular user with permissions to run Apache service

By default, all the Anchor services use local built-in SYSTEM account and on Windows, no built-in accounts except NETWORKING SERVICE have access to networking resources, where we're going to store the assets. So to be able to set everything up, you need to create a regular user (local or domain one) which we will use for Apache servers.

It should have a limited administrator permissions and be able to log in on the Apache servers and run services. Modify the Apache Service login (in Windows Services management find the Apache Service, right-click on it, select Properties -> Log On -> Log on as) on the web nodes to use this new user instead of the local SYSTEM login.

After that, modify the filesystem permissions for the Apache installation folder (typically C:\Apache24) and Anchor Server installation folder (typically C:\Anchor Server) and other locations, like the certificates folder if you store these in separate location, to allow this user a full access (read, write and execute) to these folders and all the contents inside.

Step 2: Create a networking share for assets storage

Create a regular networking share (for example `\\server\shared_assets_data`) on any of the servers you see fit, most obvious location would be the first existing Apache server. Make sure disk on which share folder will be physically located have enough space for growth, at least 20GB recommended.

Modify the permissions for this share so that the regular or domain user which you have create on the previous step, will have full access (read, write, modify) to the contents of this shared folder.

Verify access to this shared folder by trying to access it from all the deployed Apache servers using the user, created on previous step. Make sure this user will be able to create and delete files from this share from every deployed Apache server.

Step 3: Move assets to shared storage

From the current production Apache server, copy all the contents of local assets folder, typically located in:

`C:\Anchor Server\web\greensnake\static\assets`

To this newly created networking share. It's better to do it during maintenance hours when service is offline to not have any files locked or modified in there at the given time.

After copy is finished, purge the local contents of this folder (`C:\Anchor Server\web\greensnake\static\assets`) on all the Apache servers.



Note: For symbolic links used in this section to work, system drive of your Apache server should have NTFS file system and Windows version should be not earlier than Windows 7 (Windows Server 2008 R2)

Now you need to map the networking share with assets to this local filesystem location on all the Apache servers, using NTFS symbolic links. It can be done using the command like this:

```
mklink /d "C:\Anchor Server\web\greensnake\static\assets" "\\server\shared_assets_data"
```

See more information on mklink command here: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/mklink>

Doing so will create direct transparent link between networking share with assets to this local filesystem folder. This means that the application accessing a link will be seamlessly redirected by the file system driver, and no special handling is needed. To users, they appear as normal directories or files.

After making symbolic link like described above on all the Apache servers, verify what this location (`C:\Anchor Server\web\greensnake\static\assets`) which is now mounted to the networking share, is writeable for the user you have created for Apache servers by creating some small file in there and deleting it.

Step 4: Verify and configure Celery and Redis services

In Anchor Server product, Redis is used as a job queue for the Anchor Celery service. Anchor Celery service on itself is a task planner, which has a set of worker sub-processes and does a lot of important tasks:

`CELERY_BEAT` - polling all the other Celeries and creating heartbeat tasks for these to confirm they're working. Also scheduling periodical (cron) tasks such as periodic wipes, etc.

`CELERY_BACKGROUND` - monitoring the results of these heartbeats and doing some additional periodical background tasks like tokens and shares expiration, billing and usage reports, daily digests, etc.

`CELERY_EP` - performing policy enforcement and root wipe jobs.

`CELERY_FSAPI` - performing jobs that involve calls to Anchor Server nodes and waiting for results, like restore, deletion, etc.

`CELERY_MIGRATION` - performing revision migration batches, if any are scheduled, and does nothing else.

`CELERY_RDB` - performing tasks related to Root Databases, thus RDB. This involves root conversions SQLite ↔ Postgres, movement of roots between Postgres DBs, root health checks.



Note: If any mentioned above functionality doesn't work properly on your cluster, the first thing to diagnose would be the Anchor Celery service. For every worker process it's writing the separate log, all of which are in `C:\Anchor Server\logs`. Logs for Celery task executors are prefixed with "tasks" and main Celery log is called `celery-service.log`. Check these if you have any issues with mentioned tasks.

Every worker above has a job queue which is maintained in Redis, and all the Celery jobs are cluster-wide, so in any given cluster you should have only ONE working instance of Redis and only ONE active instance of Anchor Celery service.

All the additional Apache servers should be configured to use these existing instances, so you need to configure Redis in such way so that all the servers have access.

To do so, you need to modify the configuration file of your Redis instance:

```
C:\Anchor Server\redis\redis.conf
```

In there on top of the file you can see the bind parameter:

bind 127.0.0.1

To allow all the servers in cluster to connect to and use this one Redis instance, you need to:

- Assign static IP address to this machine so that it doesn't change address suddenly on reboot, etc.
- Change bind parameter from 127.0.0.1 to this static IP so that Redis service will work on an external networking interface which is accessible by other servers in cluster.
- Configure Windows Firewall or any another firewall software you have installed on this server to allow local networking connections for Redis service running on this machine. Set it to allow all connections on port 6379 for all the incoming connections originating from the local network.

After doing the above, check the connectivity with this Redis service from all other Apache servers in the cluster you have deployed. To do so, use the built-in Redis distribution tool `redis-cli` on these Apache servers which is in `C:\Anchor Server\redis`

Open command console for this location, then issue the command like this to check the connection:

```
redis-cli -h <IP_address_of_your_Redis> ping
```

For example:

```
redis-cli -h 192.168.10.77 ping
```

You should get “PONG” message as a command reply. This would mean you can access Redis from this server.

Step 5: Configure web application on all Apache servers to use common resources

Now we need to configure web application to use one common Redis instance we’ve configured on step 4. For that, you need to edit need to edit the application config file:

C:\Anchor Server\web\config.py

Add the following parameters in there:

```
CACHE_REDIS_URL = 'redis://localhost:6379/0'
```

```
BROKER_URL = 'redis://localhost:6379/1'
```

```
REDBEAT_REDIS_URL = 'redis://localhost:6379/1'
```

```
CELERY_RESULT_BACKEND = 'redis://localhost:6379/2'
```

```
STATS_REDIS_URL = 'redis://localhost:6379/4'
```

Then replace “localhost” in these parameters with the real IP of the Redis instance you have configured on the previous step.

Also, while you are editing this file already, makes sense to add the following parameters in there:

```
SQLALCHEMY_POOL_SIZE = 10
```

```
SQLALCHEMY_MAX_OVERFLOW = 20
```

To increate SQL Alchemy connections pool (will allow server to process more connections).

Next, edit PORTAL_DB_* parameters, set these accordingly to your configured Portal PostgreSQL server. Or, if you have PGBouncer connection pooling configured for your SQL servers, set these accordingly to relevant PGBouncer parameters.

After that, add or modify the parameter called

```
FILE_SERVER_HTTP_URL
```

This parameter might be absent from the config by default. It has the following format:

```
FILE_SERVER_HTTP_URL = 'http://<IP_or_hostname_of_acnhor_server>:<port>'
```

For example:

```
FILE_SERVER_HTTP_URL = 'http://192.168.10.70: 2080'
```

This parameter specifies the exact Anchor Server node this Apache server will talk to when it will be retrieving revision file data, list of files in the shares, etc. The port number here is *http_port* from the Anchor server config.ini [server] section.

You need to make sure what Anchor server you will point it to can accept connections from the rest of your internal network, so make sure *http_host* parameter in the Anchor server config.ini [server] section of the appropriate server is replaced from default value *http_host=127.0.0.1* to something like *http_host=** or *http_host=<internal_local_network_interface_IP>*, for example

```
http_host=192.168.10.70
```



Note: Be careful! Under no circumstances this mentioned *http_port* on the Anchor servers can be exposed to the outside network! It must be strictly firewalled to be available only on the local network of your cluster. This communication port uses plain HTTP and has no authentication/authorization methods as it's intended for use only for inter-node communications inside the cluster.

Now, in case you have CDN for your deployment, you can configure it here too to have the benefits of CDN cache working for all Apache servers at once. To do so, add the parameters in this config to instruct the web server application to use CDN for assets:

```
FLASK_ASSETS_USE_CDN = True
```

```
CDN_DOMAIN = 'your_CDN_endpoint_URL'
```

These should be enough for CDN to work, but in case you need more refined configuration, there are additional parameters available which can be added to config:

```
CDN_DEBUG = False
```

```
CDN_HTTPS = None
```

```
CDN_TIMESTAMP = False
```

```
CDN_ENDPOINTS = ['static']
```


Extensive information on how to configure and use these is available here:
<https://github.com/paylogic/flask-cdn?tab=readme-ov-file#flask-cdn-options>

The above configuration should be enough for the Apache server to work as a web server only which will rely on other nodes such as the given Anchor server, portal PostgreSQL server and Redis to access the data. Obviously, it makes sense to point different Apache servers to different Anchor server instances to distribute the load and add more resiliency.

Step 6: Start Apache servers and add them to load balancing setup

Now you can aggregate the Apache servers you just configured into a load balancing setup. Exact setup depends on the load balancing software you're going to use and it's up to you to decide, but here are some common guidelines on load balancing setup:

1. The load balancer should prefer the last used node for new connections from the same IP address.
2. Frontend of the load balancer should have the same SSL certs as web nodes have, for the same hostname.
3. On the backend side, the load balancer can use round-robin to distribute requests between web servers but keep sticky sessions, so that multiple requests from same client will arrive to same web server.
4. The load balancer should properly set headers X-Forwarded-Port and X-Forwarded-Proto properly when forwarding data.
5. Load balancer should not cache or proxy replies from web servers, instead should directly connect the client socket to backend Apache server socket, otherwise download or upload of large files will not work correctly.

Other than that, no special configuration should be required, Anchor uses a pretty typical web server application which is easy to load balance.

PostgreSQL and Anchor connection limits

Anchor Server service requires a significant number of available SQL connections to function properly and can easily bottleneck if the limits are not set high enough. We have two separate types of connections in this context, and these have different limits.

The first type is Portal database connections, and the second type is Root database connections. From the perspective of SQL servers, all these connections are the same, but in our product Portal database works as a global locks holder for Roots, so when you're bottlenecked on the Portal database connections, this will slow down the system as a whole, not only the functioning of the web portal.

In case you have a connection pooler in place, it has its connection limits too. So, it's important to understand how these limits are applied and how to tune these up when needed.

Remember the following:

1. The connection limit for the SQL server is per-cluster. So that limit will be shared between all the databases on this SQL server.
2. Connection pooler (PGBouncer) has a total limit similar to SQL server connection limit, but in addition to that, it's possible to set connection limits per database in PGBouncer [database] settings too if you need to refine these.
3. Anchor Server has its separate limits of connections for Portal and Root databases, and while Portal connection limits are per server, Root database connection limits are per database.
4. Apache servers also work with Portal SQL Server, keep in mind that every Apache server needs at least 10 connections to Portal SQL Server.

Relevant configuration parameters

For SQL Servers:

In C:\Program Files\PostgreSQL\<version>\data\postgresql.conf

max_connections – [see parameter description here](#).

For PGBouncer:

In config.ini - *max_client_conn* and *default_pool_size* – [see parameters description here](#).

For Anchor Server:

In C:\Anchor Server\conf\config.ini

[server] section parameters:

db_connections – PostgreSQL connection pool size for the portal database. **This setting is deprecated and should not be used anymore.** If you have it in configuration, it's better to remove it in favor of newer parameters defined in [portal] section, described below.

root_db_connections - PostgreSQL connection pool size for **each root database**. Note that these are not shared between root stores in case the databases are on the same database server. In other words, the total root store connection count multiplies for each configured PostgreSQL store.

max_root_db_connections - Maximum PostgreSQL connection pool size for root databases. Recommended to set twice the *root_db_connections*.

[portal] section parameters:

default_db_connections – Default PostgreSQL connection pool size for the portal database. Intended to supersede the *db_connections* configuration setting in the server section. If not explicitly set, the *db_connections* value is used instead.

max_db_connections – Maximum PostgreSQL connection pool size for the portal database. Recommended to set twice the *default_db_connections*.

With an increasing number of clients, to attribute for increased load, it might be needed for you to increase the limits of connections for the Portal database, as by default these are quite small (set to 20 connections per node). For high-loaded deployments, it's recommended to set *default_db_connections* to at least 50 connections and *max_db_connections* to 100.

Calculating the required limits to set for PostgreSQL

For example, let's assume you have the following cluster layout:

- 1x PostgreSQL Server which holds both Portal and Root DBs
- 2x Root databases
- 3x Anchor Servers
- 2x Apache Servers

And the following configuration on your Anchor Servers:

max_root_db_connections = 20

$max_db_connections = 40$

Here you have two Root databases which are all on the same SQL server that holds the Portal database. In that case, all the connections from Anchor Servers will go to this one single SQL server, so you can calculate the number of connections per Anchor Server like the following:

$total_connections_per_node = max_db_connections + (max_root_db_connections * number_of_root_databases)$

In our example, that would be

$total_connections_per_node = 40 + (20 * 2) = 80$

You have in this situation three Anchor Servers, so these will generate around 240 ($80 * 3$) total connections to the SQL server.

In addition to these, you have two Apache Servers, which will generate around 10 connections to the Portal database each. Also, we recommend adding 10% to the total limit on PostgreSQL Server as a safety buffer.

So, in this situation, the reasonable connection limit for the SQL Server would be:

$(total_connections_anchor_servers + total_connections_apache_servers) * safety_buffer =$

$(240 + 20) * 1.1 = 286$ connections.



Note: If you have a connection pooler (PGBouncer) in place, connection limits for it should match the SQL server limits. Remember that it's possible to set connection limits per database in PGBouncer settings, so if you have Root databases on different SQL servers, all connected to cluster nodes via PGBouncer, you can adjust limits on the connection pooler accordingly by setting up connection limits per database.

Then, for another example, let's assume you have a different, more complex cluster layout:

- 1x PostgreSQL Server which holds both Portal DB
- 1x PostgreSQL Server with 4x Root databases
- 3x Anchor Servers
- 2x Apache Servers

And the following configuration on your Anchor Servers:

max_root_db_connections = 20

max_db_connections = 40

Here you have PostgreSQL Portal DB and Root DBs on separate servers, so it makes sense to calculate the numbers of connections for Anchor Servers separately for Portal DB and Root DBs:

total_portal_connections_per_node = max_db_connections

*total_root_connections_per_node = (max_root_db_connections * number_or_root_databases)*

So, here we would have:

total_portal_connections_per_node = 40

*total_root_connections_per_node = 20 * 4 (number of Root DBs) = 80*

As we have three Anchor Servers, these will generate:

*total_portal_connections_anchor_servers = 40 * 3 = 120 connections to Portal DB SQL server.*

*total_root_connections_anchor_servers = 80 * 3 = 240 connections to Root DB SQL server.*

In addition to these, you have two Apache Servers, which will generate around 10 connections to the Portal database each. And as before we recommend adding 10% to the total limit on PostgreSQL Server as a safety buffer. So, the final limits for SQL servers would be calculated like the following:

For Portal SQL Server:

*(total_portal_connections_anchor_servers + total_connections_apache_servers) * safety_buffer = (120 + 20) * 1.1 = 154 connections.*

For Root SQL Server:

*total_root_connections_anchor_servers * safety_buffer = 240 * 1.1 = 264 connections.*

So, now, if you want to add PGBouncer in this setup, you can either set `default_pool_size` to the sum of the above, $154 + 264 = 418$. Or you can set these separately for portal and root databases in the `[databases]` section of PGBouncer settings.

Finding Additional Support

Please contact us if you need additional support.

Online KB: https://help.axcient.com/en_US/176211-x360sync-administrator-faq

Call: 800-352-0248

Submit a support ticket: <http://partner.axcient.com/>