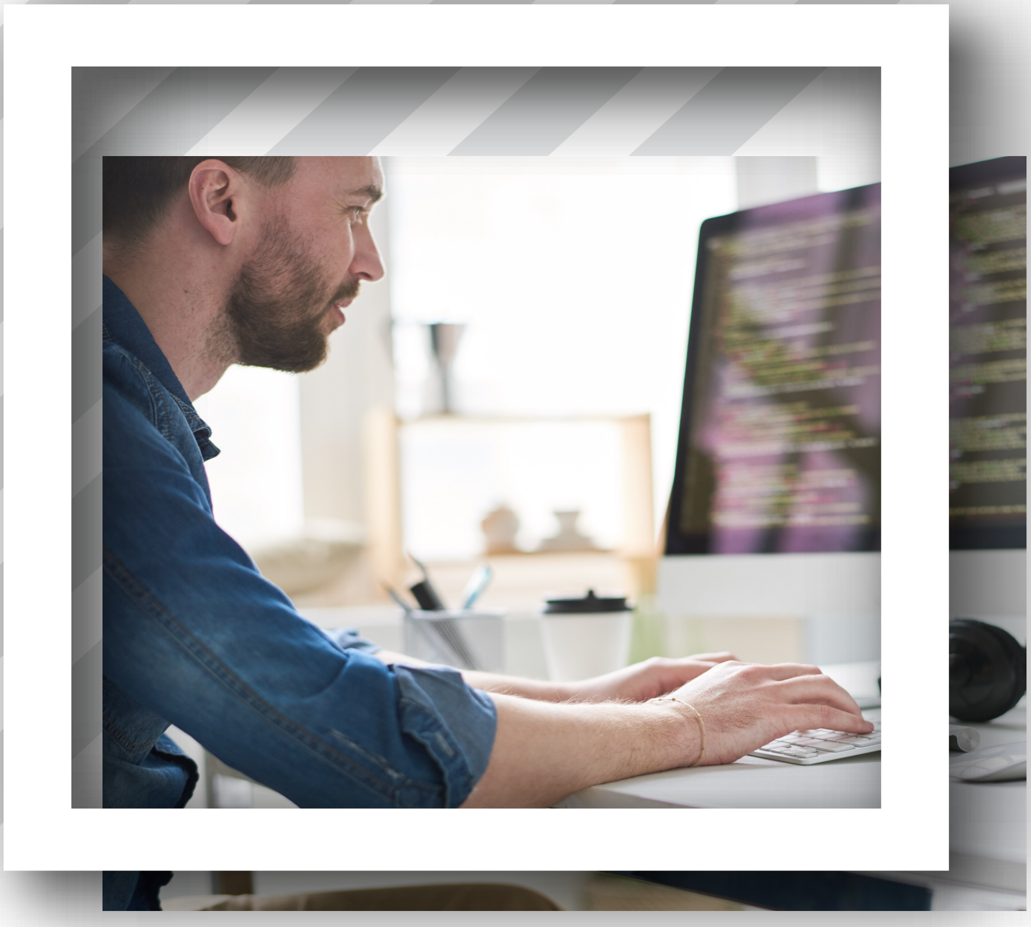
accusoft®

How PrizmDoc Viewer Load Balancing Works

Adam Cooper
Software Architect, PrizmDoc

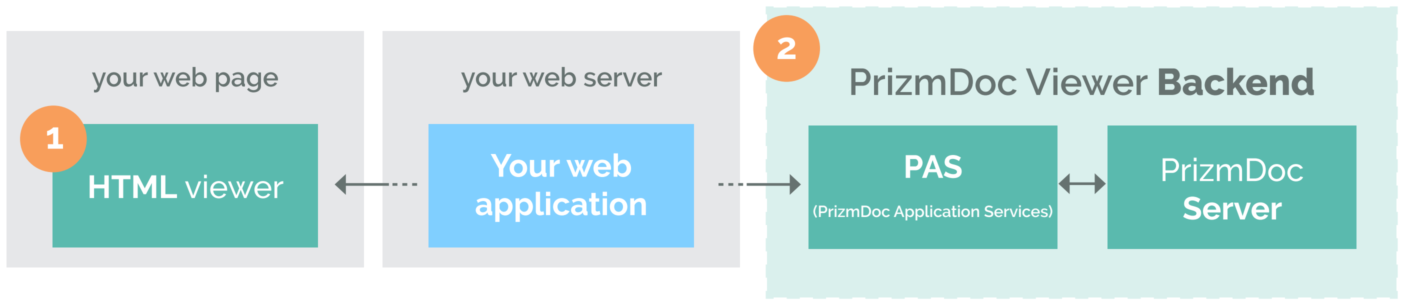


If the people using your web application need to view, search, redact, or annotate documents right in their browser, PrizmDoc Viewer is an amazing option. It lets you present Office, PDF, TIFF, email, and many other kinds of documents as part of your web application. Check out some of the demos if you've never seen it in action.

To make all of this possible, there are basically two sides to the PrizmDoc Viewer architecture:

1. The HTML **viewer** itself, running in the browser
2. A powerful **backend** which converts documents, page by page, to SVG for viewing in the browser

Your web server sits between these two, acting as a proxy for the viewer to ask the backend for the pages it needs to display:



One of the advantages of this architecture is that we can deliver the first page of the document as soon as it's ready, even while the rest of the document is still being converted. However, setting up and maintaining the backend is not trivial.

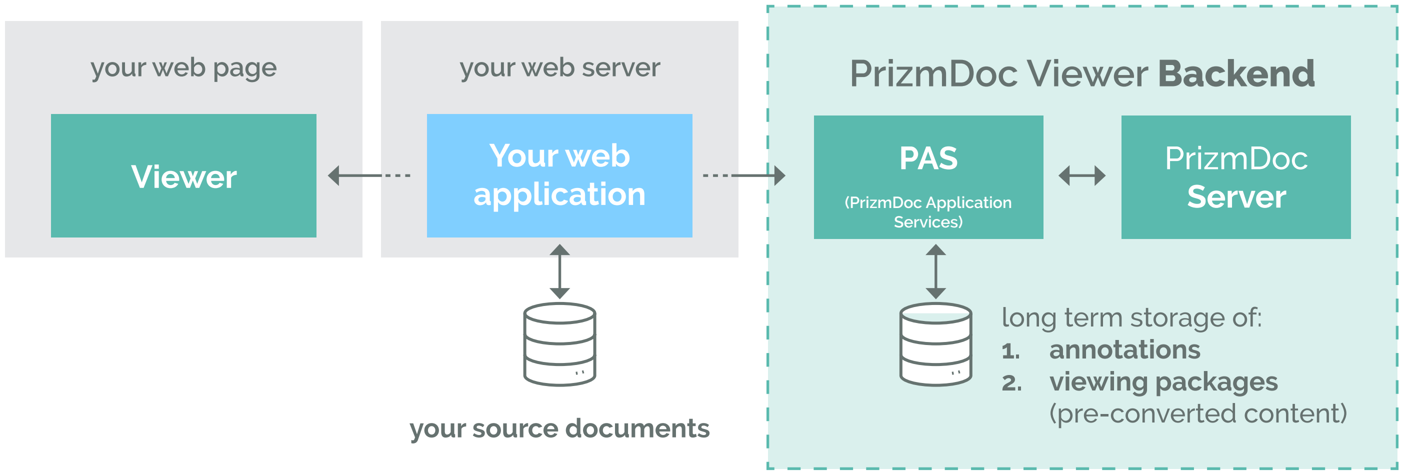
Fortunately, Accusoft can handle all of that for you with PrizmDoc Cloud. Sign up, get an API key, and simply connect your web application to our already-running, fully-managed PrizmDoc Viewer backend. It's a great option, especially if you're just getting started with PrizmDoc Viewer.

But, of course, using an Accusoft-hosted backend may not work for your business. Maybe you are not allowed to ever let documents leave your network, even temporarily. If that's the case, you'll need to host and manage the backend yourself. As customers start looking into what it takes to do that, we get a lot of questions about how load balancing works. How is the compute workload spread across the servers? How are HTTP requests routed to the correct machines? What sort of load balancer(s) should I be using? Those are the kinds of questions we'll cover in this paper.

To do that, though, we first need a more detailed picture of the backend.

What's in a PrizmDoc Viewer Backend?

Let's take a little closer look at the overall architecture, including your web application and the browser:



You'll notice that the "backend" is actually made up of two tiers: **PAS** (PrizmDoc Application Services) and **PrizmDoc Server.**

PrizmDoc Server (on the far right) is the technical heart of the product, the engine that actually converts pages of a document to SVG. It is compute intensive and has no permanent storage.

PAS (the thing that sits in front of PrizmDoc Server) doesn't actually do conversion work at all. Instead, much like your own web application, PAS has privileged access to storage that you own (say a file system or database) and delivers higher-level functionality that would otherwise have needed to be part of your web application.

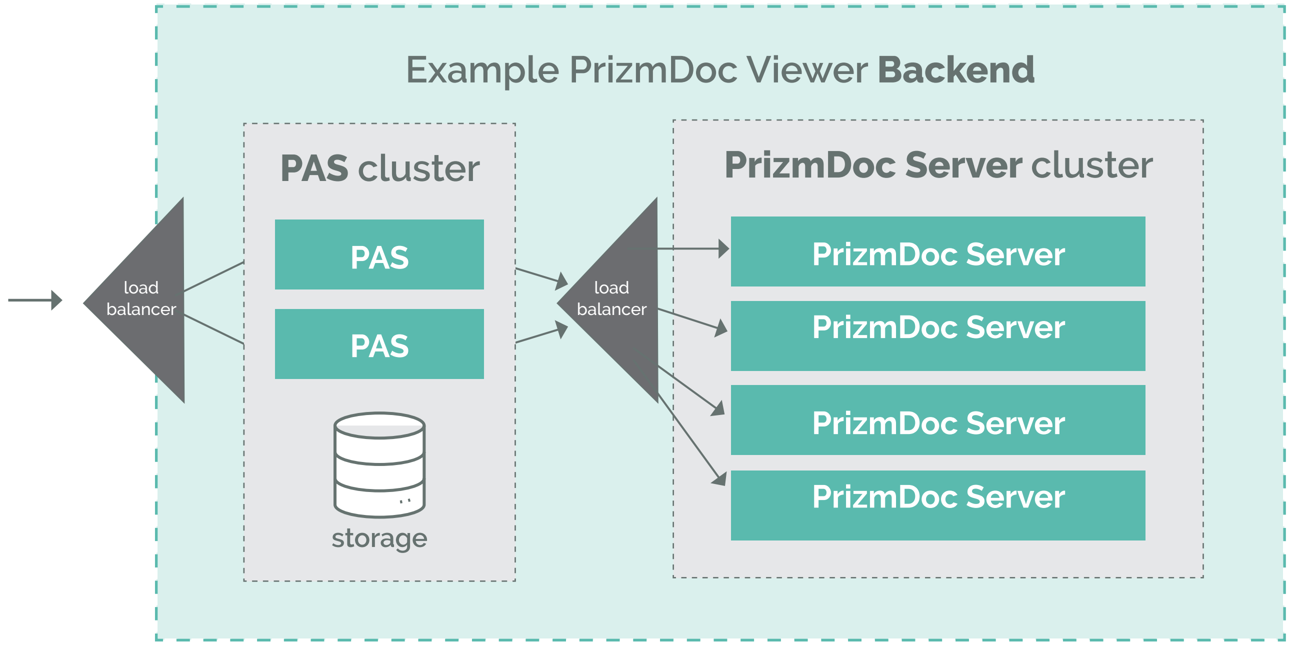
Among other things, PAS is responsible for two major features:

1. Loading and saving annotations
2. Long-term caching of pre-converted content

Conceptually, PAS is just a layer in front of PrizmDoc Server. For viewing functionality, it is the "front door" which your web application will be talking to. But, when it comes to where the actual conversion work occurs, that all happens within PrizmDoc Server.

Now, let's take an even closer look at just the backend.

For both the PAS and PrizmDoc Server tiers, you will likely[1] have multiple machines fronted by a load balancer, like so:

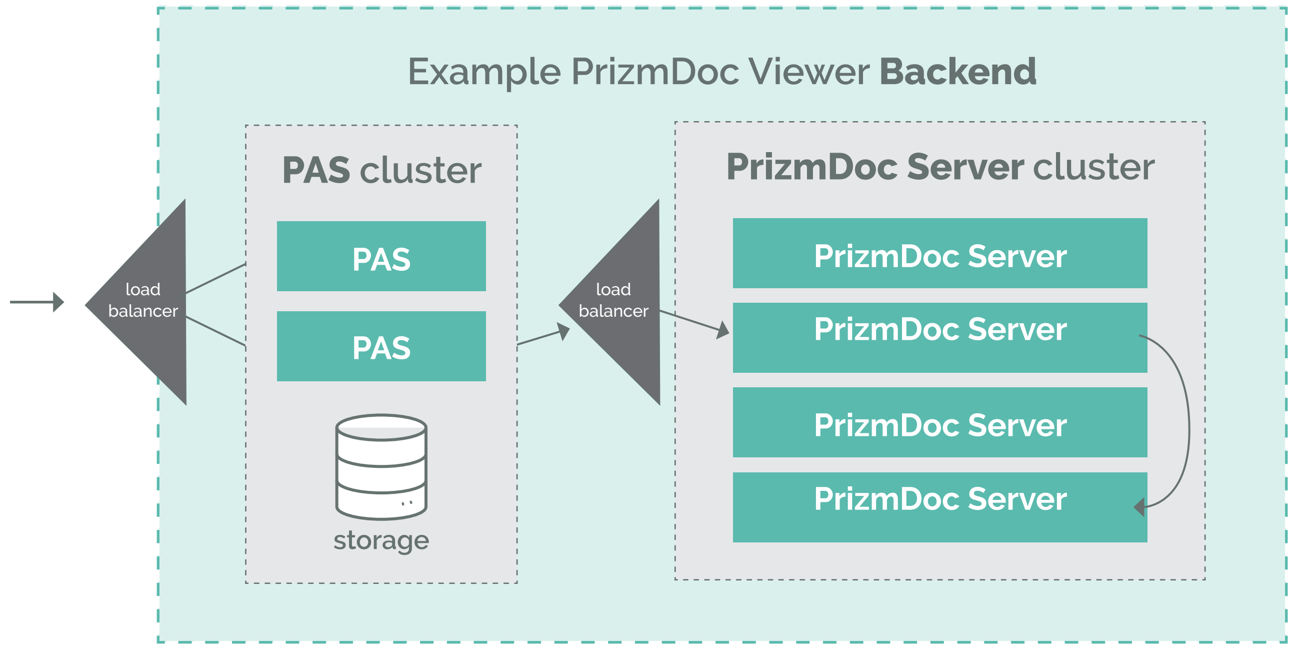


The load balancers you set up are mostly there so that each cluster has a single entry point. **Your load balancers don't need to be anything fancy; simple round-robin load balancers will work fine.** That's because the actual work of "load balancing" and request routing is ultimately handled by the PrizmDoc Server instances themselves.

How Are Viewing Requests Routed?

Whenever you start a new viewing session, all of the processing work to convert the pages to SVG is happening on one particular PrizmDoc Server machine. When the viewer sends a request like "Give me SVG for page 0," that request is piped along (through your web server, through PAS) until it gets to the PrizmDoc Server cluster, and then there is one machine, and only one machine, that can actually answer the request (PrizmDoc Server instances only use local cache; there is no shared cache or shared storage).

Fortunately, you (and your load balancer) don't need to worry about routing the request to the right machine. That's because the IP address of the correct machine is encrypted into the viewing session id itself, and every PrizmDoc Server instance knows how to pipe the request along to the correct machine if it needs to:



In fact, while we'd never recommend it, if you were to completely remove the load balancer in front of your PrizmDoc Server cluster and just send all traffic to a single PrizmDoc Server machine, everything would still get routed to the right place.

CPU Load Is Separate from HTTP Request Load

When we think about "load balancing" in the context of a PrizmDoc Server cluster, there's more involved than merely distributing HTTP requests. That's because most of the CPU work PrizmDoc Server does is background processing, completely separate from the HTTP requests it may be handling.

There are basically two kinds of HTTP requests that PrizmDoc Server handles:

1. Requests which ***start*** a new background process
2. Requests which ***get the status*** or output of a process

For viewing, your web application will make the first kind of request itself (creating a viewing session and providing the document), and then the HTML viewer will make lots of the second kind of request (getting SVG for various pages).

Every time your application creates a new viewing session and provides its associated document, PrizmDoc Server kicks off a background process to convert the document pages, one by one, to SVG (emitting pages of output as it goes). How long that process takes, and how much CPU it uses, depends entirely upon the document that is being converted.

For example, a DOCX file must first be “flowed” (paginated) before any pages can be converted, while a PDF has its pagination pre-defined. Given a DOCX and PDF with the same visible content, the PDF will convert faster and use less CPU. Also, in general, the more pages a document contains, or the more complex the content is on a page, the more CPU time it will require to fully convert.

Of course, your end users don’t have to wait for the document to be fully converted before they can start interacting with it. PrizmDoc Server delivers the first page of content as soon as it is ready, and will even proactively convert pages out of order based on what is being requested. But CPU load on the PrizmDoc Servers is not so much determined by how many HTTP requests the server is handling, but by what kinds of documents the server is converting in the background.

How Is Viewing Work Assigned to a Machine?

We’ve talked about how viewing requests get routed to the right machine, but how is viewing conversion work assigned to a machine in the first place?

The short answer is that *PrizmDoc Server will choose a machine for you by consistently hashing a unique document identifier (such as a filename, URL, or database ID) which you provide to PAS when creating the viewing session.* This ensures that the documents are evenly distributed across machines in the cluster while maximizing the chances of reusing local cache.

You see, every machine in the PrizmDoc Server cluster keeps a local cache of conversion work it has already finished. If a viewing session is being created for a document that has been viewed before, there's a huge advantage to assigning it to the same machine that converted the document last time. And that's exactly what PrizmDoc Server will try to do.

I say try because if the number of machines in the cluster changes, PrizmDoc Server may need to choose a different machine instead. Anytime a machine is added or removed, PrizmDoc Server will re-adjust how it assigns documents to machines, ensuring that documents are still being evenly distributed across the cluster. But, because we use a consistent hashing algorithm, we minimize the amount of reassignment, making it more likely than not that the same machine will still be used, even after the cluster grows or shrinks.

Whenever a viewing session is being created for a particular document, as long as the number of machines in the cluster is unchanged, PrizmDoc Server will always assign it to exactly the same machine. When the cluster size changes, PrizmDoc Server may start assigning it to a different machine but will most-likely still assign it to the same machine.

Note that, when PrizmDoc Server assigns a new viewing session to a machine, it does not take CPU load into account or try to find the machine that is most idle. Instead, it just ensures that each machine receives about the same number of documents to convert as any other. Since different documents will require different amounts of CPU time to be converted, you may see very different CPU usage patterns between machines in your cluster.

Gotcha: Accidentally Sending All Work to a Single Machine

When you create a viewing session, PAS allows you to provide the document in several ways. For example, you can:

1. Provide a URL to the document
2. Provide a path to a file which resides on the PAS server itself
3. Provide the document in a subsequent upload request

Regardless of which approach you use, PAS needs to provide PrizmDoc Server with a unique document identifier so that the viewing session can be assigned appropriately to a machine in the cluster. For most of these options, the value PAS should use is obvious, such as the URL or file path to the document.

However, there is one important case where PAS is depending on you to provide a unique document identifier, and it is probably the most common case of all: providing the document via a subsequent upload.

Take a look at this POST /ViewingSession request to PAS:

```
POST /ViewingSession
Content-Type: application/json

{
      "source": {
            "type": "upload",
            "displayName": "my-report.docx"
      }
}
```

The source object tells PAS information about how you intend to provide the original document. A type of "upload" means, "I'm going to upload the document to you in a subsequent request." The required displayName property is more important than it may seem: *displayName is the value which PAS will use as a unique document identifier, the value which PrizmDoc Server will* ***consistently hash*** *to determine which machine in the cluster the viewing session will be assigned to.* **What this means is that it is extremely important that your web application use a unique** displayName **for each document.**

Imagine what would happen if you just hard-coded your web application to use a dummy value for displayName, like "some document". PAS would give PrizmDoc Server the same document identifier every single time, and PrizmDoc Server would wind up assigning every new viewing session to only one machine in the cluster. You'd wind up with one machine doing all of the conversion work and the rest of them sitting idle.

Or, imagine a scenario where you have a bunch of different documents on your web server which happen to share the same filename. Perhaps you have hundreds of files named my-report.docx, all of them different, but all stored in different directories. What would happen if you only used the bare filename as the displayName when creating the viewing session? That's right: The viewing sessions would, once again, be artificially forced to a single machine in the cluster. In a case like this, providing just the bare filename would be a poor choice. You'd need to provide something more specific, like the filename with a path.

To avoid this trap, always make sure that, when creating a viewing session with source.type of "upload", you carefully choose a value for source.displayName. It should be a value that uniquely identifies the document you intend to upload, such as a specific-enough filename or path, or even a database identifier.

Summary

To recap:

- Accusoft fully hosts and manages a professional PrizmDoc Viewer backend as part of PrizmDoc Cloud.

- A PrizmDoc Viewer backend has two tiers: PAS and PrizmDoc Server. For viewing, your web application will communicate with PAS, but the processing will happen in PrizmDoc Server.

- You only need simple load balancers in front of your PAS and PrizmDoc Server clusters. PrizmDoc Server automatically routes viewing requests to the correct machine for you.

- CPU load on PrizmDoc Server machines is mostly determined by the kinds of documents being processed rather than by how many HTTP requests are coming in.

- PrizmDoc Server chooses which machines get new work, and it does so in a way that maximizes reuse of local machine cache (even as the number of machines in the cluster changes). PrizmDoc Server does not take CPU load into account when assigning work.

- When calling PAS to create a new viewing session and specifying that you will send the document via a subsequent upload, make sure you carefully choose a value for source. displayName. The value you pick should uniquely identify the document (like a specific file path or database identifier). If it doesn't, you could wind up forcing a bunch of work to only one machine in the cluster.

1. In general, PrizmDoc Server should run on dedicated hardware. For PAS, however, there are several options when it comes to choosing which machines you deploy to. You may choose to deploy PAS to dedicated machines, as illustrated in the diagram, or you might choose to run PAS alongside your web application, or you might even choose to run PAS on each of your PrizmDoc Server machines.