R Shiny Performance and Scalability in cloud.gov

There are a few ways to improve performance/scalability of R Shiny applications in cloud.gov, including:

1. Scale horizontally by adding instances.
2. Improve/optimize the performance of the application itself.
3. Scale vertically by adding memory/compute to each one of the instances hosting your application.
4. Add multiple listeners to each of your instances.

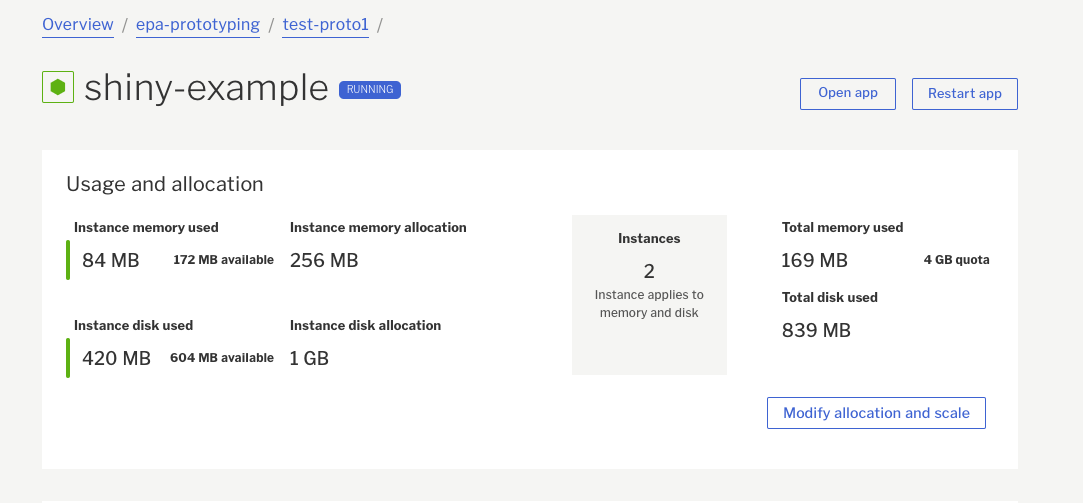
The following provides guidance on these different ways to improve performance and scalability of R Shiny applications in cloud.gov.

# Cloud.gov UI Application Configuration

Cloud.gov offers a UI interface for scaling and modifying your application. An application’s dashboard will show the usage and allocation of memory and instance disk, the number of instances running, and the total amount of memory and disk used. Figure 1 presents an example dashboard.

When ready to modify an application’s configuration for scaling, select **Modify allocation and scale**.

Figure . Shiny Example Dashboard

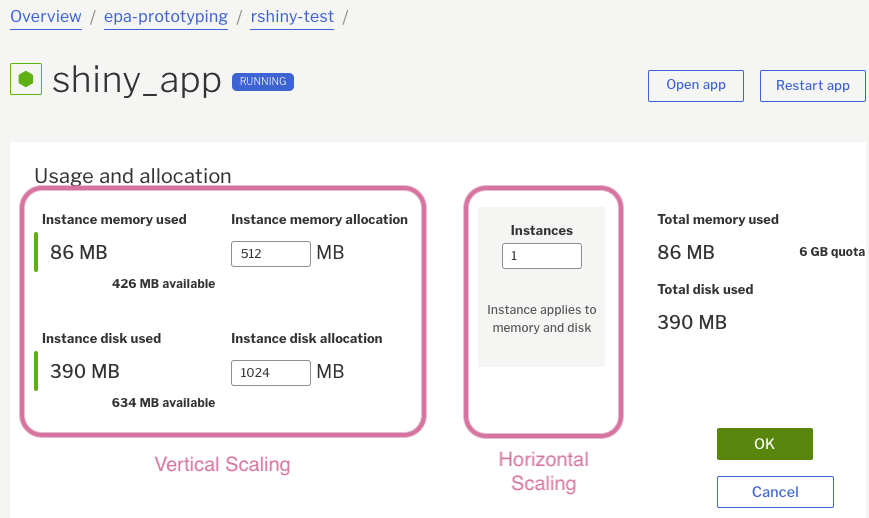


As shown in Figure 2:

* Vertical scaling configurations are controlled through the **Instance memory allocation** and **Instance disk allocation fields**.
* Horizontal scaling configurations are controlled through the **Instances** field.

Once configurations are changed, select OK to implement scaling.

Figure . Vertical and Horizonal Scaling through the Dashboard



# How can I scale my R Shiny app horizontally on cloud.gov?

Many R Shiny applications can easily scale horizontally without modification. Cloud.gov provides a simple way to scale horizontally by increasing the number of running instances for your application. The load is automatically balanced between the instances themselves. In general, a non-constrained R Shiny application can scale linearly as you add instances. So, if your application can handle 50 concurrent users with one instance, you can expect 3 instances to support up to 150 concurrent users.

If your R Shiny application performance is not constrained by access to a back-end data store or other service, use any of the following the options to increase the number of instances your application is running:

1. Use the cloud.gov dashboard to modify the number of instances. (Refer to [the UI Configuration section](#_Cloud.gov_UI_Application) for more detail.
2. Use the command line to include the number of instances you need and redeploy.
   * **To scale an existing application**, use the command:

cf scale myApp –i <x number of instances>

For example, if you want 3 instances, then use cf scale myApp –i 3

* + **To scale at deployment**, use the command: cf push myApp –i <x number of instances>

For example, if you want 5 instances, then use cf push myApp –i 5

1. Modify your manifest.yml to specify the number of instances your application requires. For example:

applications:

- name: shiny-example

memory: 256M

instances: 3

See cloud.gov’s [running multiple instances guidance](https://cloud.gov/docs/apps/multiple-instances/) for more details.

## What else should I keep in mind with horizontal scaling?

You may need to make sure that your application dependencies are not the bottleneck. For example, if your application relies heavily on a mysql database or another back-end service, the service itself must be able to handle the increased load each new instance requires. This includes processor, memory and number of connections. You will also have to scale the other services where necessary.

TENTATIVE Session management also needs to be managed <<TKTK – not clear whether cloud.gov uses sticky sessions – do they use an AWS load balancer? – see <https://cloud.gov/docs/apps/multiple-instances/> >>

# How can I optimize my R Shiny application’s performance?

If there are performance issues with your R shiny application, you will need to be able to diagnose the problem, fix the problem, and then repeat as necessary. Use the following guidance to generate load, diagnose issues and find the best strategy to fix the issues.

## To Generate Load:

Use the [shinyloadtest and shinycannon packages](https://rstudio.github.io/shinyloadtest/) to generate load on your application.

Additional guidance is available in a more rigorous [case study](https://rstudio.github.io/shinyloadtest/articles/case-study-scaling.html) where the application added a workflow to the optimization process.

## To Diagnose:

Tools such as rprof and [profvis](https://rstudio.github.io/profvis/) will help with seeing exactly where your application is spending its time.

* Rprof is R’s built-in profiling tool.
* Profvis is a graphical front-end for R’s built-in rprof profiling tool.

## To Fix:

### Plot Caching

To make your application run faster with static data, it is possible to improve the performance of individual [plots with caching](https://shiny.rstudio.com/articles/plot-caching.html).

### Promises and Async Programming

To let the end user know the application is doing something that may take a lot of time, you can improve the interactive performance of the app by [running it asynchronously](https://www.rstudio.com/resources/videos/scaling-shiny-apps-with-async-programming/).

It is possible to improve the interactive performance of the application using promises / async programming, although this is an [advanced topic](https://rstudio.github.io/promises/articles/casestudy.html).

### Other Caching

The R [memoise package](https://github.com/r-lib/memoise) can be used to cache results from function calls. If you call the same function with the same input after loading the cache, the result will be retrieved from the cache rather than re-computing the result. This library works with in-memory, file system and Amazon S3 data stores.

# How can I get the best performance out of each instance?

It is possible to increase the number of R processes running on a single instance, so you can scale your application within the instance itself to take advantage of all of its CPU and memory. According to the [scoping rules for Shiny apps](https://shiny.rstudio.com/articles/scoping.html), commercial versions of Shiny Server Pro and RStudio Connect provide functionality to run multiple R processes to achieve this.

# Other resources

[Scaling Shiny to 10,000 Concurrent Users](https://www.rstudio.com/resources/videos/scaling-shiny/)

[Shinytest – Automatied testing for Shiny apps](https://blog.rstudio.com/2018/10/18/shinytest-automated-testing-for-shiny-apps/): Performance recording and test tool referenced in 10000 Concurrent Users article

[Make Shiny fast by doing as little work as possible](https://www.rstudio.com/resources/videos/make-shiny-fast-by-doing-as-little-work-as-possible/)