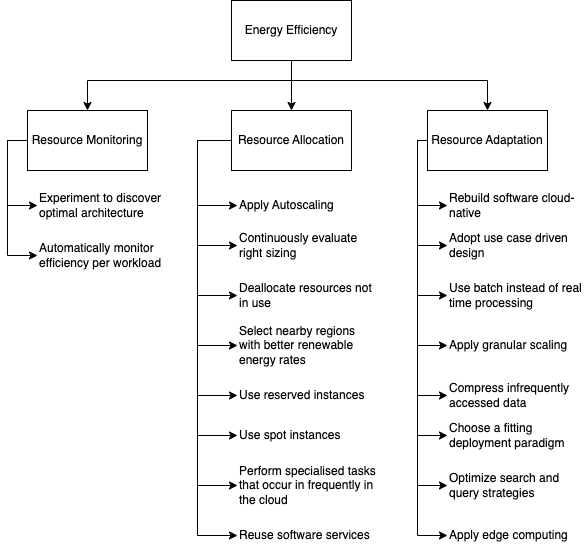
Tactic List: (From S2 Research Group / Vos et. al.)



1. Use case 1: Under-utilisation of container resources for Web Application

Q1. Which of the tactics do you consider applicable to the use case?

P2:

=> Apply granular scaling

=> Chose a fitting deployment paradigm

=> Resource monitoring

=> Apply autoscaling

=> Continuously evaluating right sizing

Q2. Have you considered using the tactic?

P2: Well, we didn’t really consider granular scaling, but it should probably lead to reduction of energy consumption.

Choosing a fitting deployment paradigm will be a great option for this use case. The workload on this web application is low and not regular. If we deploy using serverless, it should drastically reduce cost and probably the energy consumption as well.

We have considered deploying this application using the serverless deployment but it’s not a priority for the team at the moment.

Monitoring is already being done for all the applications in this platform (On cloudwatch and splunk), but in itself does not lower consumption. It can only point in the right direction to focus on.

Q3. If it was not implemented, what is the rationale for its exclusion?

P2:

Granular scaling: A general container size was provisioned for all the containers in the application at the time of deployment, so we didn’t consider granular scaling.

Autoscaling can be incorporated into granular scaling, so we would consider this more as granular scaling than just autoscaling.

The entire App-A is a container application, so it made sense to just deploy all it’s components as containers. Hence, we didn’t consider deploying this web application component as serverless at the time of deployment.

Evaluating right sizing is being done occasionally for example during cost review. This drives some of the changes the team is currently considering but not specifically for this application.

Q4. How can it be implemented in the service?

P1, P2:

In this case, we can apply granular scaling by provisioning a more fitting resource for the workload, then apply autoscaling policies to ensure it can scale up in case the traffic goes up.

To choose a fitting deployment type, we can deploy the application as serverless. This way, we don’t have to bother about scaling or managing resources. This option should be very effective for this workload.

Selection:

* Choosing a fitting deployment paradigm, considering serverless deployment is a great choice here.
* Granular scaling can be considered too, but attention should also be paid to performance of the application.

1. Use case 2: Container images are stored indefinitely

Q1. Which of the tactics do you consider applicable to the use case?

P1:

=> Deallocate resources not in use

Q2. Have you considered using the tactic?

P2: Yes, but less attention is paid to this practice. Although It is a good practice from a lean perspective. Some teams create an image on every commit. For this application that is currently the case. This makes the images pile up fast and most of these images are not being used. So deallocating the storage space of these unused containers should really be considered.

Q3. If it was not implemented, what is the rationale for its exclusion?

P2: It wasn’t intentionally excluded. But the teams don’t just really pay attention to this aspect because the cost implication is insignificant. 1GB of container storage cost only 0.1 dollars per GB per month.

Q4. How can it be implemented in the service?

P2: A good start will be to set a retention policy. The goal of the retention policy will be to set a period for which images can be stored. So images that are no longer being used can be deleted after a defined time period.

How to set the retention policy :

P2: There are images with tags like 0.0.X which are only on every commit. They could be kept for one month. The ones that are official like the 1.X.Y images should be kept longer. Let's say one year for now, but should be shorter once there is more control.

Extra notes

Q: Any specific reasons for keeping the images indefinitely?

P2: the images are stored in case of rollbacks, but only a few images actually need to be kept.

Selection:

* Deallocating unused container images after a defined time period (via retention policy is a good practice that should be considered) It will be interesting to know if it has a significant impact on energy consumption.

1. Use case 3: Gateway service (Nginx Proxy) running on large virtual machines

Q1. Which of the tactics do you consider applicable to the use case?

P2:

=> Apply autoscaling

=> Choose a fitting deployment paradigm

Q2. Have you considered using the tactic?

P2: We didn’t consider using auto scaling because of performance concerns. Also majorly because, autoscaling will be more effective if the vms are containerised. At the moment, they are not. Containerizing the vms and then scaling would require much more effort

We have considered changing the deployment method for the gateway service, to use a specialised and fully managed service which has well optimised auto scaling capability.

Q3. If it was not implemented, what is the rationale for its exclusion?

P2: We don’t consider scaling the gateway service as this might have a significant impact on the performance of the application. Also because the gateway service requires high availability, therefore scaling in or out might negatively impact the performance and availability of the Nginx Proxies

Q4. How can it be implemented in the service?

P1, P2: The team is currently working on deploying the gateway service as Api gateway. It is a fully managed, specialised AWS gateway service that leverages optimal scaling.

Api Gateway might be an effective way.

Selection:

* Choosing a fitting deployment paradigm, considering a fully managed service like Amazon Api gateway should be accessed for this use case.