Reference Architecture

Single Region

1. DNS requests can be randomly distributed across load balancers using round-robin DNS. This is a technique where multiple IP addresses are associated with a domain such that requests are distributed evenly across all of them allowing you to have multiple active load balancers. A simpler configuration, which would require no DNS configuration, would be active/standby where only one load balancer is active at a time. More complex configurations using routing protocols can also be used to ensure high availability among the load balancers.
2. Load balancers, such as HAProxy, automatically distribute requests across multiple app servers, improving fault tolerance. App servers can then also be created and destroyed on demand without impacting users.
3. The app servers are where your actual code (Ruby on Rails, Django etc.) runs. As many app servers as needed can be deployed to handle the load. Object storage should be used to store all resources and static content. The app servers should provide direct links for reading and writing rather than acting as middlemen.
4. Active/standby is the simplest configuration that helps ensure database availability. Database files should be stored on volumes as instances should be considered expendable resources that can be destroyed and recreated as needed. Backups of the volumes should also be maintained for an added layer of safety. They can be kept in object storage (as shown) or an external backup service or server.

Multi Region

1. It's highly recommended that you use a DNS provider such as CloudFlare (https://www.cloudflare.com/dns) as a way to achieve high availability when using multiple load balancers. Such services often provide a way to update DNS records almost instantly (sub minute), otherwise you may have to wait for the existing DNS record to expire (time to live). If one of the load balancers were to go down, the only way of removing it from the DNS rotation is to update the appropriate DNS record. This is where the time to live for DNS entries can be problematic.
2. Load balancers, such as HAProxy, automatically distribute requests across multiple app servers, improving fault tolerance. App servers can then also be created and destroyed on demand without impacting users. In this configuration load balancers only distribute requests across app servers within the same region.
3. The app servers are where your actual code (Ruby on Rails, Django etc.) runs. As many app servers as needed can be deployed to handle the load. If shared storage is required, solutions such as GlusterFS are available but can be complex to set up when synchronizing between regions.
4. There is one active database server in each region. If it fails, the region in its entirety should be considered sick. Standby databases can be added to each region to improve availability but this a more complex configuration. Database files should be stored on volumes as instances should be considered expendable resources that can be destroyed and recreated as needed. Backups of the volumes should also be maintained at an external location for an added layer of safety.
5. When running a database across multiple regions you must provide a way for them to communicate. Often this will require sending data across a public network. It is highly recommended that you ensure that this traffic is encrypted. One solution is to use a layer 3 tunnelling / vpn service such as tinc. It is relatively easy to use configuration management to automate the creation of tinc networks. It is important to note that adding such a service does increase the overall complexity and if the tunnel between the database nodes goes down then syncing with multi-master dbs might also fail.