I have a pretty general ‘going live’ checklist. Having done this numerous times before, here is what I would plan to do for a launch:

More than one month out (development):

1. Check that the clients (game executable running on any player machine) are not making AWS SDK calls directly to GameLift. Requests should be authenticated through other AWS infrastructure, ask us for help if this sounds like gibberish.
2. Make sure that you are carefully managing any keys or secrets used by the servers, principle of least privilege (POLP) is observed, and a rotation schedule is in place. Compromised server processes are unlikely, but such processes definitely shouldn’t have access to the keys. The best practice is to use IAM Roles to give GameLift server instances access to other AWS resources, and this should be mandatory.
3. If your server writes logs to EBS/instance store, check that these are eventually deleted (*e.g.* an hour or so later), even if they are collected and uploaded by GameLift. Use a different directory or log file name for each server process and each game session. GameLift does not manage your log file life cycle, and logs can accumulate and exhaust EBS storage on the instance. Also instances can be removed at any time causing you to lose some logs, so expect that too. A better option may be to use the Cloudwatch Log service, an ELK stack, or other log aggregation service like Splunk, LogStash or Flume. Standardize the log data so that it can be read and understood in context, in real time, and it is easy to set alarms for different failure conditions.
4. Plan the roll-out of fleets to your target regions, with queue and fleet structure. Automate the deployment process.
5. Build a load test client. Ideally, automate some level of load testing too.
6. Make sure that there is a matchmaker and a queue in more than one region. These are not redundant by themselves. The fleets can be destinations on both queues. Then if an availability zone or regions is unavailable, and you start getting errors, your back-end should fail over from one matchmaker to the other automatically. That feeling that you experience when that happens and everything continues as if nothing has happened is called operational excellence.
7. Test your system under gradually increasing loads with alpha and beta testing and soft launches (the whole checklist applies to each of these mini-launches!). Most sensitive to extremely high loads are the Game Session Placement, and Game Session Creation operations. Running longer game sessions, or reusing game sessions for automated rematching, where possible, improves these dynamics considerably. *E.g.* 100K CCU 1v1 with one minute matches matches/places/creates 833 new games per second (high) 100K CCU 4v4 with four minute matches and 2X reuse of sessions, matches/places/creates 13 new games per second (good). Spread game creation between regions. If you have unavoidably high requirements, please talk to us early on!
8. Collect lots of data! Logs and analytics are your best actionable data so think carefully about what you need to collect and how to effectively manage and analyze that data. It will drive operational management, design changes, future technical development, business intelligence, customer satisfaction and product longevity throughout the whole process.
9. Define and test the customer support experience for your players. Customers are won and lost here. Empower community managers with the tools to fix customer problems and issues.

One month out (launch readiness):

10. Select and internally communicate a launch date. Launch early in the week if possible so the first live days are work days.

11. Raise GameLift (and other) service limits to ensure that the live environment can scale up to the maximum size, which is at least the sum of the maximum instances for all fleets in the region.
12. Make sure that your dev-ops have enough visibility into your system, and effective command and control tools and protocols. Imagine that the game controls a nuclear power plant. Document (and laminate checklists for) what you think could fail, and what protocols and procedures will be used to mitigate. These protocols and procedures are sometimes referred to as runbooks. Automate the procedures if possible.
13. Make sure that your community managers are ready for the launch and hyping/exciting players through web site, social media, forums etc. in advance of your launch
14. Check that the number of servers running on an instance at full tilt (*i.e.* not just quiescent) are within the capabilities of the server instance type. Also, if there are busy activity periods during non-core gameplay, like loading a map, check that one or more games loading (typ. no players attached) doesn't impact running gameplay sessions. Lower process or thread priority from within the game server as necessary during these operations.
15. Don't let scaling policy under-provision instances for the fleet. Tune scaling policy to be more conservative at first, providing a bit more idle capacity that you think you will need. You can optimize for cost later. Consider the use of target-based scaling policy with 20% idle or more instead of rule-based policy for the best experience.
16. Check that the Flexmatch rules contain a latency rule (if you are passing latency data) so that players who are logically close to the same region typically play together. Test how this behaves under load with synthetic latency data from your load test client. If you use latency data in one part of the system then use it everywhere it can be used (Flexmatch, Game Session Queues, Backfill requests), or the results are undefined.
17. Make sure that your player authentication and infrastructure related to getting players in games (such as your own custom matchmaking) can scale effectively to meet demand. Consider load-testing this mandatory.
18. Check that the server, if left running for several days, is still able to accept connections. For example (unity users), leaving a C# TcpListener open for days will eventually stop being able to receive and respond to connections!
19. Set Cloudwatch alarms (if you are using Cloudwatch log service) for any messages that indicate severe or fatal problems on the server fleet. Simulate some failures if practical and test the runbooks.
20. Check that spot instances are typically available for the regions and instance types you are using. If, in the spot history graph, the spot prices approaches the on-demand price, this is a prime indicator that the spot instances will not be available to FleetIQ. Other historical data may be available in the AWS console.
21. Ask us ahead of time for anything you are unsure of. We are here to help and have lots of experience of developers launching.

One week out (pre-flight checks):

22. Raise your AWS support level to Business or Enterprise level, so that AWS can respond to you in the case of problems or outages outside of office hours. *Your solutions architect has no access to your fleets, and cannot help you if your servers stop working – call AWS Support first.*
23. Lock down client and server code bases completely and do a full QA suite on the final executables.
24. Establish a war room that is manned (physically and in-person if possible) by the stakeholders (e.g. developers, operations, community/support, production management, business intelligence, executive

management) for the period up to and through launch. GameLift management and solutions architects will often participate in your Slack channel during your launch window. Make sure that coffee supplies are adequate, and also do some problem solving drills using your protocols and procedures checklists/runbooks.

25. Create your live environment. Check that each queue in use has at least one on-demand fleet in each region that it must serve. Understand the rules for how queues place games (which is different depending on whether there is latency data present, or not), and create your live on-demand and FleetIQ (spot) fleets accordingly, using final executables.
26. Make sure that you use a new queue and a new matchmaking configuration for your live environment. This will ensure that you are using server code that is up to date. If you have a development or test environment that is very old (anything over a year old qualifies), it may also be worthwhile recreating these too. If you have really old fleets, you can replace those too in the process. Now all the server code is the latest and greatest.
27. Practice updates to the live environment servers. Be clear on your upgrade strategy, and how you will revert if you discover a problem with a new version. Consider using GameLift Aliases instead of fleet IDs in your system, (mandatory if not using GameLift game session queues).
28. Validate with a load test that fleets in your live environment are available and performant.
29. Check that the ports open on live fleets match the range of ports that could possibly be used by any server on the fleet instances, and are open to 0.0.0.0/0. i.e. if servers listen to ports in the range 38200 to 38300, these must be open to the world.
30. Check that instance types and concurrent process limits are correct.
31. Validate that the correct group of people are IAM users of the Live fleets. Restrict access to other people by means of limiting Admin privileges and access to live resources.
32. Check that RDP port 3389 and SSH port 22 are closed, or limited to a small address range reflecting the developer's public IP range(s). These ports (and any unused ports) should not be open to 0.0.0.0/0 under any circumstances.
33. Set fleet minimum sizes above zero, so that fleets can't accidentally turn their own scaling rules off during a scale-down event. Having a larger minimum gives small populations more flexibility, and these can be reduced later.
34. Set fleet maximum sizes sufficiently high to accommodate at least peak anticipated demand, including a good margin of 2X or so for unanticipated demand.
35. Set fleet protection policy to Full Protection on all live fleets so that scaling down cannot evict an active game session.

Launch day (liftoff):

35. As you are ready for players to start joining, pre-warm the fleets by scaling them up to a level that can accommodate the launch spike. The scaling rules will try to bring them down if you leave it long enough, so keep an eye on this. Disabling scaling policy in the console for warming may be warranted.
36. Monitor everything very carefully as first players arrive. Encourage the whole team to participate in the event in the war room. Celebrate victory.
37. Matchmakers, including Flexmatch, tend to do less well with small player pools and their behavior changes subtly with scale and global player distribution, so monitor particularly whether matching times are acceptable.
38. Monitor that player latency is within tolerable limits, and players are getting a good experience.

After launch (flight operations):

39. Tune scaling rules so that you are able to minimize idle capacity, whilst at the same time supplying all players with games.
40. Measure the latency for your players, and modify matching rules or roll-out in additional regions based on the worst cases first.
41. Listen to your players experiences, and plan changes with them in mind. Community managers are your vanguard.
42. Optimize the server executable, as its performance efficiency has a direct bearing on the fleet costs. Increasing the number of server processes per instance is a good goal.
43. Use that analytics data to drive continued development, improving player experience, game longevity and optimize monetization (in that order), and also start A/B testing.

That should get you started, thank you!